SIGRAD 2006

The Annual SIGRAD Conference

Special Theme: Computer Games

November 22-23, 2006

Skövde, Sweden

Conference Proceedings

Organized by

Svenska Föreningen för Grafisk Databehandling

and

Högskolan i Skövde

Edited by

Henrik Gustavsson

Published for

Svenska Föreningen för Grafisk Databehandling

(SIGRAD)

by Linköping University Electronic Press

Linköping, Sweden, 2006

The publishers will keep this document online on the Internet - or its possible replacement – for a period of 25 years from the date of publication barring exceptional circumstances.
The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/.

Table of Contents

Research / Work in Progress Papers

Sketches / Short Papers

Application Papers

Preface

These proceedings contain the papers from the SIGRAD 2006 conference which was held on the 22rd and 23th of November in Skövde, Sweden. The topic of this year's conference is Computer Games. As in previous years, we also welcome paper submissions in various other graphics areas.

The SIGRAD conference has an explicit ambition to broaden its geographic scope beyond the national borders of Sweden. We are therefore very happy to have several international contributions
this year. The keynote speakers this year are Torbjörn Söderman and Johan Andersson from DICE. The topic of the keynote is the history of rendering in computer games and the rendering technology used in the current generation of game engines.

We would also like to thank the program committee that provided timely reviews, and helped in selecting the papers for these roceedings.

Many thanks to our generous sponsors: Högskolan i Skövde, DICE , Pearson Education, InGaMe Lab and Akademibokhandeln.

We wish all participants a stimulating conference, and hope they take the chance and to create new connections in the Nordic graphics community.

Henrik Gustavsson
Program Chair SIGRAD 2006

SIGRAD 2006

The SIGRAD 2005 program committee consisted of experts in the field of computer graphics and visualization from Sweden. We thank them for their comments and reviews.

Program Chair

Henrik Gustavsson, Högskolan i Skövde

Program Committee Members

Henrik Engström, University of Skövde
Craig Lindley, Gotland University
Gustav Taxén, Royal Institute of Technology
Stefan Seipel , University of Gävle
Anders Hast, University of Gävle
Lars Kjelldahl, Royal Institute of Technology
Ulf Assarsson, Chalmers University of Technology
Kai-Mikael Jää-Aro
Lennart Ohlsson, Lund University

SIGRAD Board for 2006

Kai-Mikael Jää-Aro, Chair
Stefan Seipel, Vice Chair
Lars Kjelldahl, Treasurer
Anders Hast, Secretary

Anders Backman, Member
Anders Ynnerman, Member
Thomas Larsson, Substitute
Ken Museth, Substitute
Lennart Ohlsson, Substitute
Örjan Vretblad, Substitute

EFFICIENT RENDERING OF MULTIPLE REFRACTIONS
AND REFLECTIONS IN NATURAL OBJECTS

Stefan Seipel
Department of Information Technology, Uppsala
University and Department of Mathematics and Computer
Science, University of Gävle
ssl@hig.se

Anders Nivfors
Department of Information Technology, Uppsala
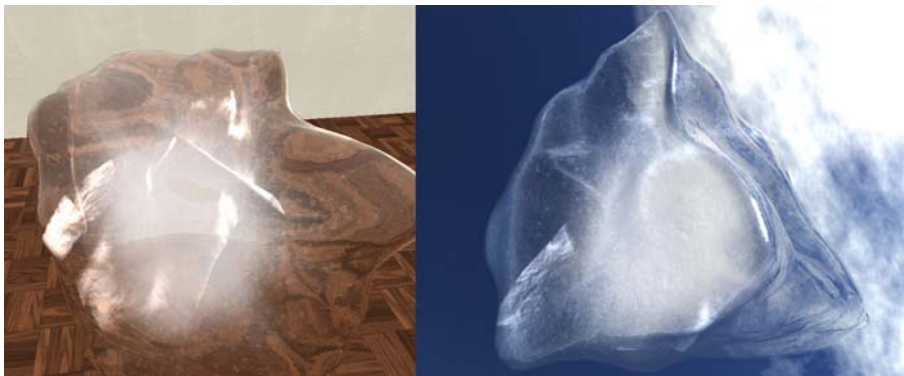University
nivfors@gmail.com

Figure 1: Rendering of ice showing multiple specular reflections and refraction

Abstract

In this paper we present a multi-pass rendering approach for approximating the effects of multiple refractions and specular reflections in transparent or semitransparent materials. These optical effects are typically found in natural materials like ice but also in glass artworks. The rendering technique proposed in this paper is intended to perform at real-time frame rates and aim at achieving naturally looking effects rather than simulating physically correct interaction between light and matter. Part of our method is an improved image space technique for clipping a geometry using the Boolean difference of two geometries in order to create internal surfaces of reflection inside transparent objects. It employs a number of generic cracks surface geometries which are clipped against the geometry to be rendered. Reflection and refraction effects on the ice are implemented by using environment mapping. Two-sided refraction is accomplished by combining the normal vectors of the front and back side of the ice object. Our method renders icy objects with convincing visual appearance in real-time on state-of-the-art graphics hardware.

Keywords: Computer Graphics, Modelling of Natural Phenomena, Illumination Models, Realtime Rendering Algorithms.

1. Introduction

Objects made of glass or naturally grown ice are impressive to look at. Part of the fascination these materials exert on the observer is due to the rich and extreme interaction effects between light and glass or ice objects. Ice shows, in comparison with glass, an even more complex visual appearance because its internal structure is often much more irregular and inhomogeniuos as opposed to glass. The simulation of these complex interactions between light and ice or glass comes almost for free in raytracing approaches. In the field of real-time computer graphics, however, rendering of ice has not been subject of intensive research.

Realistic rendering of natural phenomena in real-time has always been one of the most difficult tasks. In consequence, numerous papers can be found that describe implementation techniques for fire, smoke, clouds, fog, water etc. Yet, rendering of ice appears to be little explored. In this paper we summarize the most prominent visual characteristics of ice and what distinguishes ice from similar materials such as glass. We then present a multi-pass rendering approach to accomplish these characteristics in real-time using the GPU of a modern graphics card.

2. Related Work

To our knowledge, until now hardly any research has been published in relation to real-time rendering of ice objects. Little of the work related to ice in the field of computer graphics is mostly concerned with physically based techniques for offline rendering.

A physically based model for icicle formation was presented as early as 1993 [Kharitonsky and Gonczarowski 1993]. Methods for ice crystal formations have been looked into quite thorough [Kim and Lin 2003; Kim et al. 2004; Witten and Sander 19891]. Other work focusing at animations of melting materials, such as ice, has been presented by Carlson et al. [Carlson et al 2002] and Jones [Jones 2003].

Whereas most of the above mentioned papers describe offline methods which focus on the formation of ice, Kim and Lin [Kim and Lin 2003] presented techniques for ice formation and rendering that achieves interactive frame rates for low resolution models.

In the work we present below, we focus on visual appearance of icy objects rather than physically correct modeling. Therefore, in order to achieve real-time performance, we aim at techniques that utilize the graphics hardware. In particular we address tricks for rendering refraction and reflection effects found in ice structures.

Since ray tracing still can not be carried out in real-time, interactive graphics applications (i.e. games) generally use environment mapping (EM) [Blinn and Newell 1976], in order to collect light samples from in the nearby environment of an object to be rendered. EM is hardware supported, straight-forward to use and very fast. Environment mapping does, unfortunately, not support recursive reflections nor does it account for backside normal contribution.

In the paper "Interactive Refraction on Complex Static Geometry using Spherical Harmonics" [Génevaux et al 2006] Génevaux et al present a method that achieves realistic two-sided refraction. The method pre-computes some light paths which are used for approximations of the refracted paths during rendering. Since it relies on a pre-computation step, the method is suitable for static objects only. The runtime computations can be calculated on the graphics hardware, which allows the method to perform at interactive frame rates.

Another approach to approximation of two sided reflection was presented by Wyman [Wyman 2005]. By saving the normal vectors for the back facing polygons of the object to a texture along with the thickness of the object, both the back and front normal of the object can be found and accounted for when refracting the incident ray.

Khan [Khan 2004] presents an interesting method to achieve two-sided refraction when using EM by saving a normal cube map for the object. When rendering the surface using a shader, instead of sampling the environment map immediately, the normal cube map is sampled to obtain the normal for the backside of the object. With these two normal vectors the incident ray can be refracted two times hence accounting for both the back and front side of the object. The method suffers from artifacts in the refracted image. Being based on EM, neither Wyman's nor Kahn's method can handle concave objects very well.

Figure 2. Photograph of an ice-block frozen in a refrigerator.

3. Visualizing features of ice objects

3.1 The look of ice

What renders the visual appearance of icy objects specific is a combination of several different optical effects which are due to the internal morphology and current surface properties of ice. The surface of icy objects can exhibit very different specular reflection properties. For instance, melting ice has a highly specular surface which gives rise to total reflections. In contrast, under cold conditions, the surface of ice is populated with thousands of microscopically small ice crystals yielding to a rather rough diffuse reflection when viewed from a distance. Ice is a semi-transparent material and its light transmission properties do change extremely depending on the internal structure of ice. This is further complicated due to the fact that the morphology of ice changes depending on external conditions like temperature and pressure. Obviously, there is no general model to describe the visual appearance of ice in all its possible different states. It is likewise difficult to recreate the visual appearance of ice by explicitly modeling all the microstructures in ice that give rise to the various visual appearances. Still, a reasonable approach to realistic rendering of ice is to identify a number of typical properties which tell the observer that it is ice she is looking at as well as to apply a number of simplified rendering techniques that approximate the most prominent visual attributes. Figure 2 shows a real ice object and, without any doubt, we can tell that we look at ice rather than e.g. glass. The most prominent features which we in informal interviews with six students identified to contribute most to the visual appearance of ice are:

a) Transmission of light from behind the object involving double refraction at front-face and back-face.

b) Specular reflections on the outer surface combined with specular highlights occurring at interior surfaces (cracks).

c) Irregular shape and bumpiness of the surface.

d) Air enclosed inside the ice causes milky white appearance.

This list of visual features is not meant to be all comprehensive, and not all of these characteristics must necessarily be found in ice at the same time. Yet, an integration of these four attributes into a material renderer should deliver a plausible visual result for most natural ice objects.

Methods for rendering of the visual properties mentioned under c) above are readily available under the term bump-mapping in computer graphics text books. Air bubbles and internal light scattering as mentioned under d) above, however, pose some bigger problem if we intend to implement them in a general way. In the course of our work, we have previously been using texture impostors. Hereby, we randomly place a number of transparently textured polygons inside the ice object. The texture images depict air bubbles at certain sizes and spatial frequencies. During a separate rendering pass, these texture images are rasterized and composited using alpha-blending. In this report, we do not describe the technique in more detail; instead we refer to [Nivfors 2006].

3.2 A straightforward approach to multiple refraction

In real-time computer graphics, environment mapping (EM) is used to accomplish effects of reflections and refractions. In our approach we use a cubic environment map that is sampled in the fragment shader of the ice object using the refracted and reflected eye vectors. In order to account for motion of the ice object or any nearby object in the scene, the environment map is updated dynamically if so necessary. The EM texture is updated by rendering the entire scene six times with a field of view set to 90°, rotating the camera to point in all six directions from the centre of the ice object.

When a ray penetrates a transparent object, the ray is refracted at two surfaces: when the ray enters the object and when the ray exits the object. In standard EM only the surface orientation (normal) of the front face is used for refracting the ray. To achieve two-sided refraction we use a simplified method similar to the one presented by Wyman [Wyman 2005]. We render the normal vectors of the back faces of the ice object to a texture using a separate rendering pass. Hereby, the vertex shader calculates the world space normal and the fragment shader normalizes and interpolates the result into a back-face normal buffer. A floating point texture is used for the result to avoid precision artefacts. The normal texture for the back-faces of the object is later sampled by the ice's fragment shader, allowing to simply blend the back and front faces' normal vectors using equation 1.

$$\mathbf{n}_r = \mathbf{n}_f(1-a) - \mathbf{n}_b(a) \qquad (1)$$

where $\mathbf{n}_r$ is the resulting normal
$\mathbf{n}_f$ is the front face's normal in world space
$\mathbf{n}_b$ is the back face's normal in world space, sampled from a texture
a is a scalar value in the range of [0, 1] indicating how much of the back face's normal that should be applied. A value of 0.33 is used for a in the screenshots in this report.

Using $\mathbf{n}_r$ as surface normal, the eye vector (e) is refracted using equation 2. The eye vector, or view vector, is a vector from the viewer to the vertex on the surface and is calculated in the ice object's vertex shader. The interpolated result is used in the fragment shader.

$$\mathbf{r}_{refr} = eta \cdot \mathbf{e} - \left(eta(\mathbf{n}_r \bullet \mathbf{e}) + \sqrt{1 - eta^2\left(1 - (\mathbf{n}_r \bullet \mathbf{e})^2\right)} \right) \cdot \mathbf{n}_r \quad (2)$$

where $\mathbf{r}_{refr}$ is the refracted eye vector
eta is the refraction index of the media we are travelling from divided by the media we are travelling to. In this case air/ice = 1.0003/1.3091 ≈ 0.7641 [Nordling and Österman 1999].

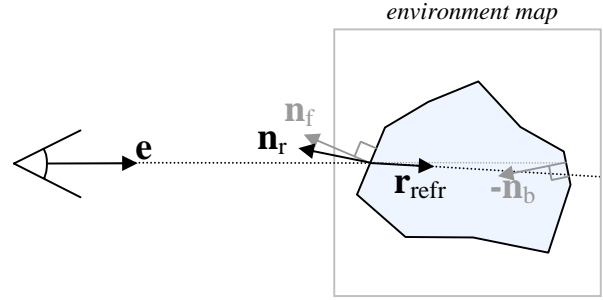$\mathbf{r}_{refr}$ is then used to sample the environment map. Figure 3 shows the concept of the method.

Figure 3. The idea behind two sided refraction using blended normal vectors.
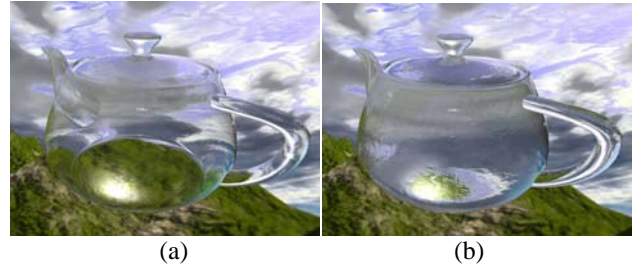

| (a) | (b) |

Figure 4. Model of the teapot with the ice shader demonstrating (a) two-sided refraction using blended normals compared with (b) one-sided refraction. Air and cracks are turned off.

So instead of following the incident ray and refracting it twice, a new normal is simply created by combining the back and front face's normal vectors. This is not a physically correct method but since ice is a highly distorted material this is a reasonable trade off for increased performance. Also, highly concave objects will not refract correctly since only one front face and one back face are accounted for.

For the purpose of reflection calculations, only the normal vector of the front face needs to be accounted for. So e is simply reflected using equation 3.

$$\mathbf{r}_{refl} = \mathbf{e} - 2(\mathbf{n}_f \bullet \mathbf{e}) \cdot \mathbf{n}_f \qquad (3)$$

where $\mathbf{r}_{refl}$ is the reflected eye vector.

Reflected and refracted light is sampled by using the reflection and refraction vectors as indexes into the environmental map. The resulting two color samples are then mixed using an approximation of the Fresnel equation similar to the one proposed in [Jensen and Golias 2001]. The Fresnel equation determines the ratio between the reflected and refracted ray's color contribution using a rather heavy equation. Our approximation is created with the intention of being fast and yet to produce a convincing visual result rather than being physically correct (equation 4).

$$f = \left(1 - |\mathbf{e} \bullet \mathbf{n}|\right)^3 \qquad (4)$$

where f is the Fresnel term
$\mathbf{e}$ is the eye vector to the vertex
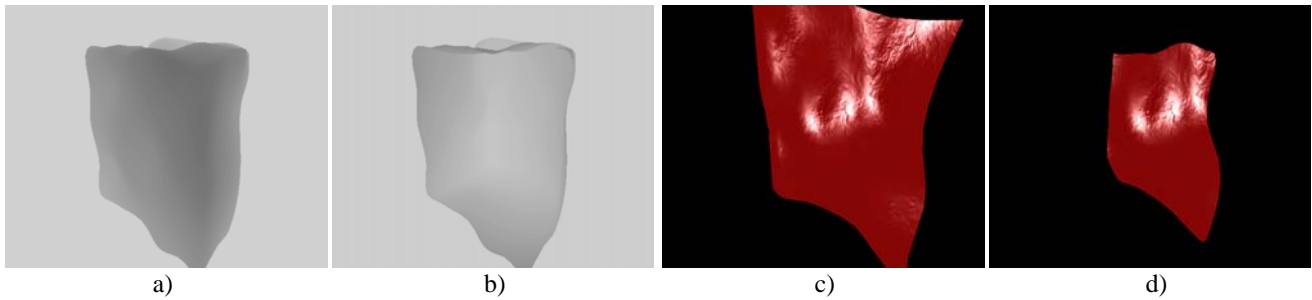$\mathbf{n}$ is the vertex's normal.

Figure 5. The depth buffer saved to a texture for the (a) front faces and (b) back faces of the ice object. (c) The crack model without clipping. (d) The final clipped crack. The cracks are colored red on these pictures for better visibility.

Equation 4 is calculated in the ice object's vertex shader and f is interpolated and sent to the fragment shader where it is used for mixing the reflection and refraction colors sampled from the environment map (equation 6). The result is shown in Figure 4.

3.3 Multiple specular reflections

The occurrence of cracking surfaces inside solid ice object is very typical for natural ice objects. It is at these internal, often curved and irregularly shaped crack surfaces where incident light is reflected towards the observer. An example of these internal specular reflections can be seen in the left part of a real ice block in figure 2. To accomplish this light contribution due to internal specular reflections, we use a rendering pass that only calculates a specular highlight for internal crack surfaces and which uses bump mapping for crack surfaces contained inside an ice object. Hence, if no specular highlighting occurs, the crack is completely transparent. Crack surfaces are rendered without back face culling to make them visible from both sides as an ice object is rotated. So, how do cracks appear inside ice objects?

Unless they are explicitly defined by the model creator, we need to define a means of adding a certain number of cracking surfaces into the object. Generally, it would be preferable to add an arbitrary number of cracks dynamically to any ice model without explicitly defining them as subcomponents of the object geometry. We therefore propose to use a number of generic crack models of sufficient size, to be subsequently used for rendering of any ice model. These crack models can be positioned and randomly oriented inside an ice model. Knowing the bounding box of an ice object, the scale of the generic crack geometries can easily be adjusted to cover the volume of the object. The difficulty lies in performing a volumetric clipping operation, that prevents crack surfaces from exceeding the ice objects actual boundaries.

Constructive Solid Geometry (CSG) [Goldfeather 1986] is a way of constructing solids by combining existing solids using the Boolean operations such as union, difference and intersection. Various techniques of performing interactive CSG using the default depth buffer of the fixed point rendering pipeline exist already (e.g. [McReynolds and Blythe 1996]). With the possibilities provided by programmable graphics cards, we make use of a more straightforward screen space method. It uses a render-to-texture approach for both frontal and rear faces and employs a specific fragment shader which evaluates depth values of the crack geometries in order to perform interactive CSG.

Consequently, cracks are created at rendertime by first randomizing the rotation and position, preferably near the centre of the ice. Then for each frame the depth values of the back and front faces of the ice model are rendered to a texture (Figure 5 (a, b)). In order to optimize the precision of the depth values, the near plane of the viewing frustum is aligned with the front most distance of the object's bounding volume and the far plane is aligned with the farthest distance of the object's bounding volume. Then the cracks can be rendered as usual with their shader that performs bump mapping and specular highlighting. In the fragment shader some instructions are added to sample the two depth textures and to compare the current pixel's depth value with the corresponding depth interval. If the value is not contained within the interval of the two sampled values nothing is drawn since then the pixel is outside the ice object. Figure 5 (c, d) shows a generic crack geometry rendered without and with clipping against the objects boundary. Regardless of the number of crack surfaces, only three render-to-texture passes need to be performed: the ice object's front and back faces' depth values and the result with the clipped cracks.

This method is based in the assumption that the object that should be clipped (the crack) is a surface model and not a solid. Furthermore, only Boolean intersection is performed. So the method can not be considered a full alternative method for interactive CSG, although it can probably be altered to be more general. The method only works correct for convex objects, otherwise artefacts will show. The method fulfils the criteria of an appropriate clipping method for the cracks and can handle an arbitrary amount of cracks.

4. Performance

We implemented an application to demonstrate and evaluate the proposed method using C++/OpenGL. For all passes that render to a texture the framebuffer object (FBO) extension was used [Juliano and Sandmel 2006]. When using FBOs the color or depth values can be written directly to a texture instead of being copied from the frame or depth buffer. Shader programs were written in Cg (version 1.4.1).

The resolution of all the textures attached to the FBOs (for the depth buffers, crack bumps, and air bubbles) was set to 512x512 for the measured benchmarks as well as in the screenshots in this report. The only exception was the texture keeping the backface normal vectors which was set to the same resolution as the application window, 1024x768, to avoid filtering artefacts of the normal vectors. In all tests the ice covered approximately 1/5 of the screen. Unless stated otherwise, in all tests all effects were enabled, with 19 air planes and 2 cracks rendered and clipped in real-time using our method.
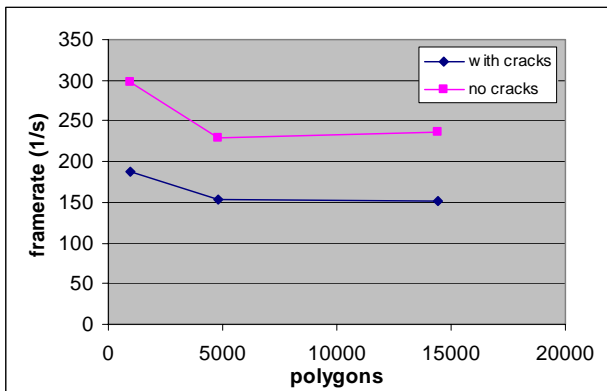
Figure 6. Frame rates for increasingly complex objects rendered without cracks and air-textures (upper curve) or with cracks and air-textures (lower curve), respectively.

For the performance tests we used an AMD Athlon 64 3700+ 2.2 GHz with 1 GB of RAM; the tested graphics card was nVidia's GeForce 6600GT with 128 MB of video memory. We measured framerates for three different objects: A sphere with 960 polygons, an irregular ice-cube with 4800 polygons, and the teapot counting 14400 polygons. Figure 6 illustrates obtained framerates for a static scene.
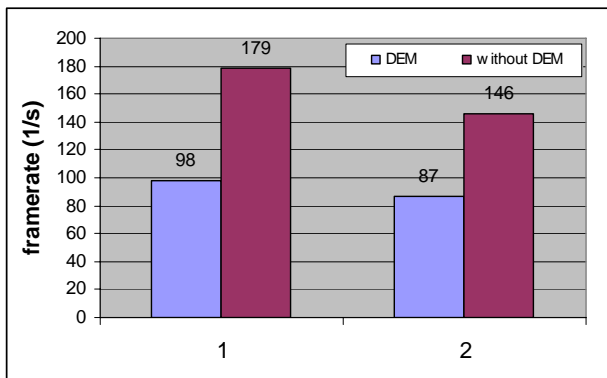

Figure 7. Frame rates without two-sided refraction (category=1) and with the approximation for two-sided enabled (category=2).The blue bars indicate rendering performance with dynamic update of the environment cube map. Purple bars are for a static environment map.

We tested how rendering performance is affected when our approximation of two-sided refraction based on normal vector blending is used. To that end, the ice cube object (4800 polygons) seen in figure 1 was rendered in a simple landscape environment containing 2340 polygons. It was rendered in a dynamic scene which requires a continuous update of the environment cube map. For comparison, we rendered the scene also without dynamic update of the environment cube map i.e. a static map. The results of these measurements are shown in figure 7.

5. Discussion and conclusions

In the final implementation of our ice renderer, the proposed rendering method uses many rendering passes and the ice shader accesses no less than six texture maps including the environment cube map. Yet, this did not affect the performance too much. This is partly due to the use of frame buffer objects but also thanks to the very powerful GPU that performs all calculations. Figure 6 shows that high frame rates are achieved even with fairly complex ice objects. The teapot has three times as many faces as the ice cube but the frame rate is barely affected, which suggests that the rendering method is fill bound. Apart from air bubbles, one of the most important features in our renderer is cracks in the ice. When using the methods proposed in this paper the frame rate drops about 26% compared with rendering without air and cracks. The performance mainly depends on two factors: how many pixels the ice object occupies on screen and the resolution of the textures attached to the FBOs. If the ice covers approximately 1/5 of the screen 148 fps is achieved, compared to 67 fps when the ice fills the entire screen. This is because the fragment program for ice is complex and requires many texture accesses. While this might be seen as a problem for large icy objects, it makes on the other hand distant or small pieces of ice very cheap to render.

Our clipping method fulfils the goal of a method for adding an arbitrary amount of cracks to any convex object. However, a possible future improvement would be to implement some pre-computation step which clips the cracks once and saves them as new models. In that way, cracks would not have to be clipped for each rendered frame yielding increased performance. The method could even be constructed to handle concave objects. When testing our ice rendering programs, we experimented with letting the cracks affect the refracted image. We did this by modifying the method for two-sided refraction in such way, that the crack normal vectors would also distort the final surface normal. To that end cracks were rendered on top of the ice object's back faces when the backside normal texture was rendered. The cracks seemed to affect the final result a bit too much, especially when many cracks were used. Still, in nature the cracks do affect the refracted image, so if pre-computed CSG is to be implemented a modified version of this technique might prove useful. One small drawback of pre-computing the cracks is that if the ice is to be animated, e.g. due to melting, the computation of new vertices for the cracks in real-time would be too expensive.

In regard to our approximation of two-sided refraction using normal vector blending, the observed frame rates shown in figure 7 demonstrate that the extra performance penalty is, with approximately only 10%, relatively low.

In our rendering model, cracks and air in the ice are coarsely approximated rather than being based on physical correct models. Yet, the visual result is convincing, in particular for animated scenes. The pictures obtained (see e.g. figure 1) show typical characteristics of ice at high frame rates using standard hardware. This makes the method suitable for interactive applications such as computer games.

We have presented a comprehensive method for real-time rendering of ice incorporating multiple refraction and reflection on internal surfaces. Given a more or less convex geometry, it creates an ice object filled with air particles and bubbles. The number of cracks added to the ice can be specified in real-time. The environment and light sources are dynamically reflected by the bumpy surface, and a refracted environment can be seen through the ice. Our objective was to recreate visual effects of ice in real-time. The demonstrated results show the relevance of these

visual characteristics, was well as the obtained frame rates that confirm real-time performance on standard graphics hardware.

References

BLINN, J. F., and NEWELL, M. E. 1976. Texture and Reflection in Computer Generated Images. Communications of the ACM v. 19 i. 10, ACM Press, New York, 542-547.

CARLSON, M., MUCHA, P. J., VAN HORN III, R. B., and TURK, G. 2002. Melting and Flowing. In Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation, ACM Press, New York, 167-174.

GÉNEVAUX, O., LARUE, F., and DISCHLER, J. 2006. Interactive Refraction on Complex Static Geometry using Spherical Harmonics. In Proceedings of the 2006 symposium on Interactive 3D graphics and games, ACM Press, New York, 145-152.

GOLDFEATHER, J. 1986. Fast Constructive Solid Geometry Display in the Pixel-Powers Graphics System. In Proceedings of the 13th annual conference on Computer graphics and interactive techniques, ACM Press, New York, 107-116.

JENSEN, L. S., and GOLIAS, R. 2001. Deep-Water Animation and Rendering. Gamasutra article. http://www.gamasutra.com, April 2006.

JONES, M. W. 2003. Melting Objects. In Journal of WSCG 2003, 247–254.

JULIANO, J., and SANDMEL, J. (contact persons). 2006. Framebuffer object extension specification. http://oss.sgi.com/projects/ogl-sample/registry/EXT/framebuffer_object.txt, May 2006.

KHAN, A. 2004. Dual-sided Refraction Simulation. Shadertech Contest, Summer 2004. http://www.shadertech.com/contest, April 2006.

KHARITONSKY, D., and GONCZAROWSKI, J. 1993. Physically based model for icicle growth. The Visual Computer: International Journal of Computer Graphics, Springer-Verlag New York Inc., Secaucus, 88-100.

KIM, T., and LIN, M. C. 2003. Visual Simulation of Ice Crystal Growth. In Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation, Eurographics Association, Aire-la-Ville, 86-97.

KIM, T., HENSON, M., and LIN, M. C. 2004. A hybrid Algorithm for Modeling Ice Formation. In Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation, ACM Press, New York, 305-314.

MCREYNOLDS, T., and BLYTHE, D. 1996. Programming with OpenGL: Advanced Rendering. Course notes from ACM SIGGRAPH 1996 Course 23, ACM Press, New York, 31-42.

NIVFORS, A. 2006. Real-time rendering of ice. Masters Project Thesis in Computer Science. Department of information Technology, Uppsala University.

NORDLING, C., and ÖSTERMAN, J. 1999. Physics handbook for Science and Engineering. Studentlitteratur.

WITTEN, T. A., and SANDER, L. M. 1981. Diffusion-limited aggregation, a kinetic critical phenomenon. Physical Review Letters 47, 1400–1403.

WYMAN, C. 2005. An Approximate Image-Space Approach for Interactive Refraction. In Proceedings of ACM SIGGRAPH 2005, ACM Press, New York, 1050-1053.

Distributed Ray Tracing In An Open Source Environment
(Work In Progress)

Gunnar Johansson[*]
Linköping University

Ola Nilsson[†]
Linköping University

Andreas Söderström[‡]
Linköping University

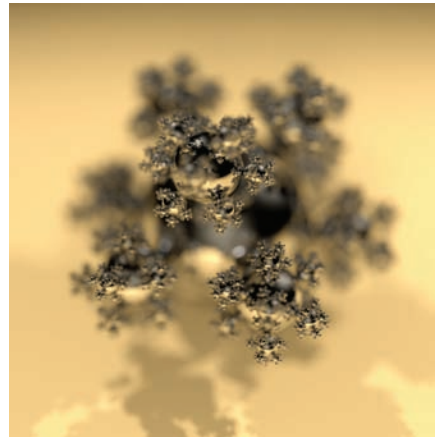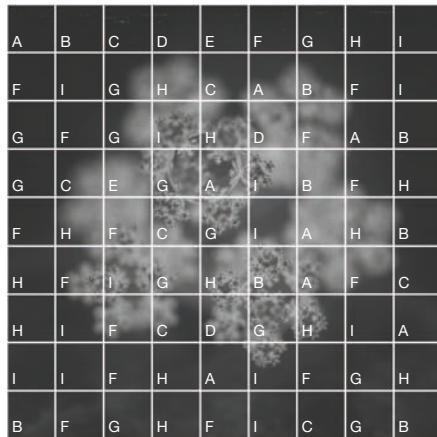Ken Museth[§]
Linköping University

Figure 1: Network distributed rendering of a scene with simulated depth of field. Left: The partitioning of the scene and the node assignment is shown, the grey scale denotes complexity of the rendering measured in time spent per pixel. Right: The ray traced final image, note that a large amount of samples per pixel are needed to sufficiently sample this scene (512 rays per pixel). Below: Bar-chart depicting the load balancing measured in number of buckets rendered: A: Athlon XP 2.1GHz, B: P4 1.7GHz, C: Celeron 2.8GHz, D: PIII 700MHz, E: PIII 450MHz, F: Athlon MP 2.1GHz, G: Athlon MP 2.1GHz, H: PPC 970 2.5 GHz, I: Opteron 2.4 GHz.

Abstract

We present work in progress on concurrent ray tracing with distributed computers using "off-the-shelf" open source software. While there exists numerous open source ray tracers, very few offer support for state-of-the-art concurrent computing. However, it is a well known fact that ray tracing is computationally intensive and yet prevails as the preferred algorithm for photorealistic rendering. Thus, the current work is driven by a desire for a simple programming strategy (or recipe) that allows pre-existing ray tracing code to be parallelized on a heterogenous cluster of available office computers - strictly using open source components. Simplicity, stability, efficiency and modularity are the driving forces for this engineering project, and as such we do not claim any novel research contributions. However, we stress that this project grew out of a real-world need for a render cluster in our research group, and consequently our solutions have a significant practical value. In fact some of our results show a close to optimal speedup when considering the relative performances of each node. In this systems paper we aim at sharing these solutions and experiences with other members of the graphics community.

CR Categories: I.3.2 [Computer Graphics]: Graphic Systems—Distributed/Network Graphics D.1.3 [Programming Techniques]: Concurrent Programming—Distributed Programming

Keywords: distributed ray tracing, render farm, open source

[*]e-mail: gunjo@itn.liu.se

[†]e-mail: olani@itn.liu.se

[‡]e-mail: andso@itn.liu.se

[§]e-mail:kenmu@itn.liu.se

1 Previous work

Rendering of images lies at the very heart of computer graphics and the increased desire for photorealism often creates a computational bottleneck in image production pipelines. As such the subfield of global illumination has been the focus of numerous previous publications, most of which are beyond the scope of this paper. To mention some of the most important; Ray tracing [Appel 1968; Whitted 1980], beam tracing [Heckbert and Hanrahan 1984], cone tracing [Amanatides 1984], radiosity [Goral et al. 1984], path tracing [Kajiya 1986], metropolis light transport [Veach and Guibas 1997], and photon mapping [Jensen 1996]. While some of these techniques offer better performance than others a prevailing problem seems to be that they are computationally intensive. In this paper we have chosen to focus on the popular ray tracing technique extended with photon mapping. However, even this relatively fast method can crawl to a halt when effects like depth of field or caustics are added. Unlike the usual pin-hole camera, realistic camera models usually require a significant increase in the amount of samples. The same is true when ray tracing caustics of translucent materials. Especially the latter has been a major issue in our group, since we are conducting research on large-scale fluid animations.

The algorithms constituting ray tracing and photon mapping are of-

ten referred to as being "embarrassingly parallel". Nevertheless a large body of work has been devoted to this topic [Lefer 1993; Freisleben et al. 1997; Stone 1998; Lee and Lim 2001], but relatively little has found its way into open source systems. *Yafray*[1] has some seemingly unstable support for multi-threading, but currently no distributed rendering capabilities. *POV-Ray*[2] has some support for distributed computing through unofficial patches. Finally, for *Blender*[3] some unmaintained patches and utilities appear to provide basic network rendering. PBRT[4] is another ray tracer; without distribution capabilites but with excellent documentation and modular code design. Given the state of the aforementioned distributed systems we choose - in the spirit of simplicity and modularity - to work with PBRT as opposed to competing ray tracers. We then extend PBRT with distribution capabilities which is straightforward given its modular design.

2 System

Two fundamentally different strategies exist for concurrent computing. The first approach is *threading* which involves spawning multiple local threads (i.e. "lightweight processes") sharing the same execution and memory space. In the context of ray tracing these threads can then cast rays in parallel. The second strategy is to employ *message passing*, which provides parallelism by communication between several running processes each having their individual execution and memory space. The former is more efficient when considering shared memory architectures, but is obviously not extendable to clusters of computers. For this reason, we use a message passing technique which has optimized strategies for both shared memory systems and clusters.

2.1 Software and Implementation

We have developed a modular distributed rendering system based on simple modifications of pre-existing open source components. The system can be configured to run on almost any computer hardware using very little effort, creating a powerful rendering cluster at virtually no cost. The core component of our system is the "physically based ray tracer", *PBRT*, by Pharr and Humpreys [2004]. We have extended PBRT with OpenMPI [Gabriel et al. 2004] and LAM/MPI [Burns et al. 1994; Squyres and Lumsdaine 2003] to support concurrent computations on both shared memory and distributed architectures. Brief descriptions of these core components follow:

- PBRT is open source for non-commercial use and offers an advanced light simulation environment. It deploys a strictly modular design and the source code is well documented.

- MPI (Message Passing Interface) [Forum 1994] is the de facto standard for distributed scientific computing. The implementations we used are OpenMPI and LAM/MPI which offer full compliancy to the MPI-2 standard.

To simplify the administration of our cluster, we use the *warewulf cluster solution*[5]. The warewulf server makes it possible for nodes to boot off the network while automatically downloading a specialized Linux configuration. Using this solution, virtually any machine connected to the network can be rebooted into the cluster and

---

[1]http://www.yafray.org

[2]http://www.povray.org

[3]http://www.blender3d.org

[4]http://www.pbrt.org

[5]http://www.warewulf-cluster.org

automatically register with the server. In addition, it is easy to maintain an up-to-date installation across the nodes and poll CPU and memory usage.

Following the modular design of PBRT, our implementation is based on dynamically loaded modules. This simplifies the network abstraction layer significantly. PBRT is executed using the MPI runtime environment on every node in the network. Then, each node in turn loads different modules depending on its role. With this setup we have implemented a star shaped rendering cluster which has one master node that is connected to all the render nodes, see figure 2. The purpose of the master node is to distribute work, collect data, and assemble the final image. Since the complexity of different parts in the image is not known prior to rendering, the work distribution is a non-trivial problem, which is further complicated by the heterogeneity of the network itself. We chose to implement an automatic load balancing scheme where nodes query the master for tasks as the rendering progresses. First, the master partitions the rendering work into buckets, for example blocks of pixels or sets of samples. Then, each rendering node is assigned a bucket in sequential order. As soon as a node completes rendering, the finished bucket is sent to the master node for compositing. If there are unfinished buckets, the master returns a new bucket assignment to the render node and the process loops. This simple scheme is summarized in algorithm 1 and 2 for the master and render nodes, respectively. This strategy of dynamic task assignment is expected to automatically balance the load between nodes of different performance, providing good utilization of the assigned render nodes.
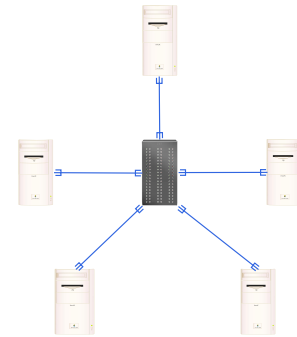
Figure 2: We use a simple star shaped network layout. The master node is shaded black and the render nodes white.

1: Compute the buckets for distribution
2: Assign initial buckets to the nodes in sequential order

3: **while** there are unfinished buckets **do**
4:    Wait for a bucket
5:    Add received bucket to image
6:    **if** there are unassigned buckets **then**
7:       Assign the next unassigned bucket to sending node
8:    **else**
9:       Send terminating signal to sending node
10:    **end if**
11: **end while**

Algorithm 1: RENDERMASTER()

In practice, the implementation consists of a few lines of changes in the PBRT core files and additional modules that implement our distribution strategies outlined in algorithm 1 and 2. Our implementation currently supports distribution of either:

```
 1:  Initialize sampler and preprocess the scene

 2:  Wait for initial bucket assignment
 3:  while node is assigned a bucket do
 4:      while bucket is unfinished do
 5:          Fetch a sample from sampler and trace ray
 6:          Add sample result to bucket
 7:      end while

 8:      Send finished bucket to master
 9:      Wait for new bucket assignment or termination signal
10:  end while
```

Algorithm 2: RENDERNODE()

Pixel blocks consisting of $\{R,G,B,A,weight\}$ tuples.

Sets of samples consisting of $\{x,y,R,G,B,A\}$ tuples.

Photons consisting of light samples $\{x,y,z,R,G,B,n_x,n_y,n_z\}$.

where capital letters denotes the usual color channels, lower case is used for coordinates and n_i denotes normal vector components. When distributing blocks of pixels, the render nodes will return a block of finalized pixels to the master upon completion. This leads to low, fixed bandwidth demands that are proportional to the block size. However, this also requires that the nodes sample in a padded region around the block for correct filtering of the boundary pixels. This means more samples in total, introducing an overhead growing with smaller block sizes. If we assume that the block is quadratic, with side l, and the filter has a filtering radius of r, then the overhead is given by $4rl + 4r^2$. In practice this means that a block size of 100×100 together with a typical filter radius of 2 already gives an overhead of $4 \times 2 \times 100 + 4 \times 2^2 = 816$ pixels out of 10.000, or roughly 8 % per block. This overhead needs to be carefully weighted against the better load balancing achieved by smaller block sizes. For example a block size of 10×10 pixels with filter radius 2 introduces an overhead of 96 % per block.

Adversely, the distribution of samples does not introduce a similar computational overhead, since the samples are independent of each other. However, this approach leads to higher bandwidth demands. More specifically, the bandwidth required for sample distribution compared with pixel distribution is proportional to the number of samples per pixel. For some scenes this can be a considerable number (see for example figure 1).

The two previous distribution strategies deals exclusively with the casting of rays within the ray tracer. However, the celebrated photon-map extension is also easily distributed, with nearly the exact same code. We have parallelized the pre-proccessing step which samples light at discrete points in the scene; the light estimation can be parallelized using one of the two aforementioned modules.

2.2 Hardware

Currently, our cluster is composed of existing desktop computers used by the members of our group, in addition to a collection of very old retired computers collected in our department. The performance ranges from a Pentium III 450 MHz to a dual core AMD Opteron 280, giving a very heterogeneous cluster. The majority of the machines are connected through 100 Mbps network cards and a few are equipped with 1000 Mbps cards. Finally, to better evaluate the impact of the very inhomogeneous hardware we cross-test on a strictly homogeneous shared memory system with high network bandwidth and low latency; an SGI Prism with 8 Itanium II 1.5GHz CPUs, 16 GB of RAM and InfiniBand interconnects.

3 Benchmarks

To assess the performance of our render systems we have benchmarked several different tests. By varying both the raytraced scene and the bucket distribution we evaluate the load balancing. Figure 3 shows the result of applying the load balancing scheme as described in algorithm 1 and 2 for both pixel and sample distribution. The pixel distribution strategy is very close to optimal, and clearly outperforms the sample distribution approach. Our tests indicate that this is due to the currently inefficient, but general, computation of the sample positions.

To measure the speedup achieved by the cluster, we measured the rendering times of a benchmark scene at low resolution at each render node. This gives a relative performance measure for the nodes that can be accumulated for a given cluster configuration. Figure 4 shows the speedup achieved by the cluster for respectively pixel and sample distribution, and figure 5 shows one of the corresponding load distributions. Since we can configure the cluster with an arbitrary combination of nodes, we chose to generate four different host-lists and plot the speedups achieved by increasing the number of nodes from each list. As can be surmised from Figure 4, the pixel distribution shows a clear advantage over the sample distributions. This is in agreement with the results shown in Figure 3. However, the pixel distribution deviates from the optimal linear speedup when the cluster reaches an accumulated relative performance of 5-6 units. This illustrates the main weakness of our simple load balancing scheme. Currently, the master assigns buckets to the render nodes in a "first-come-first-served" manner. Consequently it is possible that a slow node is assigned one of the last buckets, making the final rendering time depend on the execution of this single slow render node. Our measurements indeed show that fast nodes typically spend a significant amount of time idle, waiting for the final buckets to complete. A better strategy for load distribution or re-distribution is clearly needed to improve performance. This is left for future work, but we have several promising ideas.

Figure 6 shows a frame from one of our gigantic fluid animations (a water fountain) placed in the "Cornell-box". The fluid interface is represented by a sparse level set data structure and directly raytraced. Note that the complex fluid surface requires a very large amount of recursive steps in the ray tracing to accurately account for the light transmittance. The lighting conditions are very demanding due to a high amount of indirect lighting and color bleeding. The scene is rendered in a resolution of 1600×1600 with photon mapping using the final gather step. We stress that both the ray tracing and photon shooting scale well even for a scene as complex as this.

Our final benchmark is presented in Figure 7. It shows how the rendering times are affected by the number of buckets for a given scene. As expected, the pixel distribution strategy suffers from a large overhead induced by many small buckets, while the rendering times using a sample distribution approach is left almost unaffected. Also note the initial local minima in rendering time for the pixel distribution, caused by the good load balancing resulting from an optimal bucket size.

4 Future Work and Conclusions

As can be seen from figure 7 the input parameters to our system greatly affect performance. A practical extension would be a friendly user interface to the not always intuitive parameter space. The user should only be concerned with adding a scene description to a queue. For this we need to design heuristics or algorithms that from a given set of parameters {image size, scene complexity, cluster size, cluster node performance, *etc.*} finds the optimal settings and forwards an augmented scene description to the system.
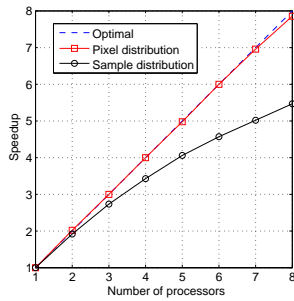
Figure 3: Benchmarking the scene in figure 1 at resolution 900×900 with 16 samples per pixel partitioned using 81 buckets. The rendering is performed on an SGI Prism shared memory machine with 8 CPUs. Note how the speedup is close to optimal for pixel distribution, while the sample distribution is suffering from overhead in the computation of the sample positions.
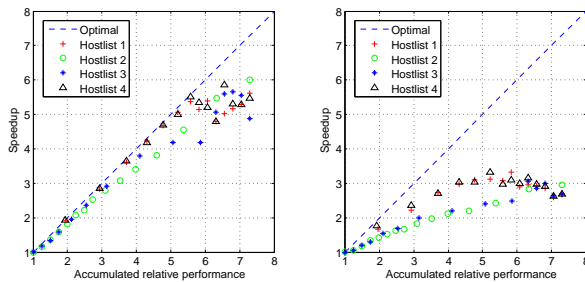

Figure 4: Benchmarking the scene in figure 1 at resolution 900×900 with 16 samples per pixel partitioned using 81 buckets. The rendering is performed on a heterogeneous cluster of 15 nodes. Left: Pixel distribution. Right Sample distribution.

Handling errors arising from computations in a distributed environment is a difficult problem. Implementations of MPI such as FT-MPI [Dewolfs et al. 2006] are completely dedicated to building stable systems. For a normal application such as ours the capabilities of OpenMPI are adequate but still require careful design. We wish to make our system as stable as possible.

Any parallel computation is bound to asymptotically saturate the network when using the layout described in 2. So far we have not seen this, but the amount of network traffic for distributing samples is considerable already at 10-20 machines. Thanks to our modular design it is easy to implement different layouts. This will also spread some of the computational overhead for the master node. We attribute the surprisingly bad results for sample distribution (figure 4 right) to the relatively expensive sample position computations. This can be efficiently solved by specialized routines for each sampling pattern. As can be seen from the load balancing graphs, figure 1 and 5 with a complex scene and a heterogenous cluster it is difficult to get really good utilization. Our simple scheme is stable and works asymptotically well (at least for sample distribution) but has practical drawbacks. We want to benchmark balancing schemes with better real world performance such as

- *"Round robin"* network style where each node only communicates with its neighbors and share its current task. This should decrease the network traffic to specific master nodes and distribute the workload better.
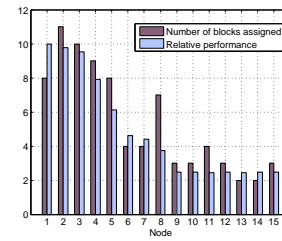

Figure 5: The load balance from the pixel distribution (figure 4 left) using a random host file.

- *"Optimal work load assignment"*, currently each work unit is equally large. Using a preprocessing step where the time complexity of the scene is measured, as well as the relative performance of the nodes, a better work load assignment should be feasible. Stability issues when dealing with assumptions on performance and complexity needs to be considered though.

- *"Maximum utilization"*, modify the current load balancing scheme so that when idle nodes are detected the distributed work units are recursively split into smaller parts and reassigned. This should ensure good utilization at a small overhead.

In this work, we have described how to use existing open source components to compose a high performance cluster used for distributed ray tracing with little effort and low budget. Initially, we have tested a simple load balancing strategy based on a first-come-first-served scheme. The tests indicate that for homogeneous configurations the resulting speedup is close to optimal. However, for heterogeneous scenes and cluster configurations sub-optimal results were reported stressing a need for more sophisticated strategies, as outlined above.

References

AMANATIDES, J. 1984. Ray tracing with cones. In *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 129–135.

APPEL, A. 1968. Some techniques for shading machine renderings of solids. In *AFIPS 1968 Spring Joint Computer Conf.*, vol. 32, 37–45.

BURNS, G., DAOUD, R., AND VAIGL, J. 1994. LAM: An Open Cluster Environment for MPI. In *Proceedings of Supercomputing Symposium*, 379–386.

DEWOLFS, D., BROECKHOVE, J., SUNDERAM, V., AND FAGG, G. 2006. Ft-mpi, fault-tolerant metacomputing and generic name services: A case study. *Lecture Notes in Computer Science 4192*, 133–140.

FORUM, M. P. I. 1994. MPI: A message-passing interface standard. Tech. Rep. UT-CS-94-230.

FREISLEBEN, B., HARTMANN, D., AND KIELMANN, T. 1997. Parallel raytracing: A case study on partitioning and scheduling on workstation clusters. In *Hawai'i International Conference on System Sciences (HICSS-30)*, vol. 1, 596–605.

GABRIEL, E., FAGG, G. E., BOSILCA, G., ANGSKUN, T., DONGARRA, J. J., SQUYRES, J. M., SAHAY, V., KAMBADUR, P.,
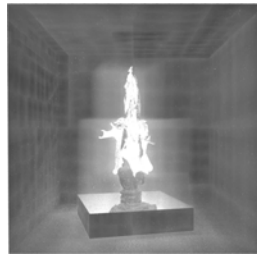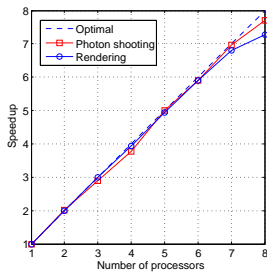
Figure 7: Benchmarking pixel distribution versus sample distribution with an increasing number of buckets.

Figure 6: Above: A complex scene from a water simulation inserted into the "*cornell box*" containing several difficult components. Below left: The speedup from distributing the photon mapping preprocessing step and the ray tracing. Below right: Time complexity visualization of the scene measured in time spent per pixel, bright pixels indicate complex regions.

BARRETT, B., LUMSDAINE, A., CASTAIN, R. H., DANIEL, D. J., GRAHAM, R. L., AND WOODALL, T. S. 2004. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, 97–104.

GORAL, C. M., TORRANCE, K. E., GREENBERG, D. P., AND BATTAILE, B. 1984. Modeling the interaction of light between diffuse surfaces. In *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 213–222.

HECKBERT, P. S., AND HANRAHAN, P. 1984. Beam tracing polygonal objects. In *Computer Graphics (SIGGRAPH '84 Proceedings)*, H. Christiansen, Ed., vol. 18, 119–127.

JENSEN, H. W. 1996. Global Illumination Using Photon Maps. In *Rendering Techniques '96 (Proceedings of the Seventh Eurographics Workshop on Rendering)*, Springer-Verlag/Wien, New York, NY, 21–30.

KAJIYA, J. T. 1986. The rendering equation. In *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 143–150.

LEE, H. J., AND LIM, B. 2001. Parallel ray tracing using processor farming model. *icppw 00*, 0059.

LEFER, W. 1993. An efficient parallel ray tracing scheme for distributed memory parallel computers. In *PRS '93: Proceedings of the 1993 symposium on Parallel rendering*, ACM Press, New York, NY, USA, 77–80.

PHARR, M., AND HUMPHREYS, G. 2004. *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

SQUYRES, J. M., AND LUMSDAINE, A. 2003. A Component Architecture for LAM/MPI. In *Proceedings, 10th European PVM/MPI Users' Group Meeting*, Springer-Verlag, Venice, Italy, no. 2840 in Lecture Notes in Computer Science, 379–387.

STONE, J. E. 1998. *An Efficient Library for Parallel Ray Tracing and Animation*. Master's thesis.

VEACH, E., AND GUIBAS, L. J. 1997. Metropolis light transport. In *ACM SIGGRAPH '97*, ACM Press, 65–76.

WHITTED, T. 1980. An improved illumination model for shaded display. *Communications of the ACM 23*, 6, 255–264.

Temporal Face Normal Interpolation

Jindřich Parus [*]
Centre of Computer Graphics
and Data Visualization,
University of West Bohemia,
Pilsen, Czech Republic

Anders Hast [†]
Creative Media Lab,
University of Gävle,
Gävle, Sweden

Ivana Kolingerová [‡]
Centre of Computer Graphics
and Data Visualization,
University of West Bohemia,
Pilsen, Czech Republic

Abstract

Normals of triangular faces are essential vectors in many areas of computer graphics. In this paper we will deal with methods for normal computation of triangles under linear soft-body deformation, i.e., the triangles which deform in time so that each vertex travels independently along its linear trajectory. Linear deformation can be found in mesh morphing, cloth simulation, physical simulation, etc. We will demonstrate five different approaches for temporal face normal interpolation, one of them is new, and we will discuss their pros and cons.

1 Introduction

Normal vectors are important vectors in many areas of computer graphics. For example in shading, they are used for light model evaluation [Phong 1975][Gouraud 1971]. In rendering they are used for back face culling to exclude primitives, which can not be visible, from the rendering pipeline and thereby reducing the amount of primitives to be sent to the graphics pipeline. In computational geometry they are used, e.g., for point containment queries, to distinguish between the interior and exterior of objects. In collision detection they are used for example for collision response computation [Eberly 2004].

A normal vector $\mathbf{N}$ at a point $\mathbf{P}$ of the surface S is a vector perpendicular to the surface S at point $\mathbf{P}$. In computer graphics we usually work with some approximation of the real surface. Due to its simplicity a piecewise linear approximation is very often used to represent surfaces. Here a surface is described by a set of triangular faces. Such representation is called a mesh. It is a very widespread representation because it is very easy to modify, store and render since it is supported by graphics hardware.

In the case of meshes we usually do not define a normal in each point of the surface. Instead we define the normals of each piecewise linear segment, i.e., triangular face. Then it is supposed that the normal is constant over the whole face. The normal of the face is usually computed by taking the cross-product of two edges of the triangle. In fact we can use any two vectors which lay in the plane of the face and which are not parallel. Nonetheless since the triangular face is defined by three vertices, the easiest way to compute the face normal is to use two edges. Note that there are always two possible face normals depending on which order of vertices we choose for normal computation. It is usually required that all normals of the mesh are consistently oriented, which implies con-

sistent orientation of faces, i.e., order of vertices which define the face. Face normals are used for example in flat shading, where a light model is evaluated only once for one face, then the entire face is filled with one single color. It is of course very fast but the final image may produce Mach bands, which are disturbing.

In many graphics applications it is required to compute normals for the vertices. A usual approach for computation of a vertex normal for the vertex $\mathbf{v}$ is to take a weighted average of normals of faces which are incident to the vertex $\mathbf{v}$. There are several strategies for which weights to choose [Jin 2005] e.g., area of the faces, the angle between the edges at the vertex $\mathbf{v}$, etc. Again, face normals are essential. Normals at vertices can then be interpolated over the face resulting in smooth normal variation, which is essential in shading. So for vertex normal computation it is required to compute the face normals as well.

Many problems in computer graphics have t-variant nature, i.e., data change in time. In the case of meshes, meshes may move in space, rotate, deform, etc. Once the mesh is modified it is necessary to update the normals as well to be able to carry out operations as described above with respect to a current state of the mesh. Usually we have some static version of a certain operation and when the state of the mesh is changed the operation is recomputed according to the new state of the mesh. It is simple but in some cases it may be very slow.

The normal computation methods for t-variant meshes can be categorized into two groups according to what kind of motion the mesh performs. The first group is rigid body-motion. In the rigid-body motion the relative position of any two vertices stays fixed during the transformation and the object transforms as one entity [Karni 2004]. Examples of rigid-body motion are rotation and translation. The second group is soft-body motion. In the soft-body motion there are no restrictions on change of relative position of any two vertices; each vertex can travel along its trajectory independently of the other vertices.

The rigid-body motion can be expressed by a transformation matrix $\mathbf{A}$. Then the normal vectors of the transforming mesh under rigid-body motion is transformed by the inverse transpose of the Jacobian matrix $\mathbf{J}$ of the transformation matrix $\mathbf{A}$, thus $(\mathbf{J}^{-1})^{\mathbf{T}}$ [Gomes 1999]. Moreover, if the transformation is linear, the normal field is transformed by the matrix $\mathbf{A}$ directly, because if the transformation represented by the matrix $\mathbf{A}$ is linear then it holds that $(\mathbf{J}^{-1})^{\mathbf{T}} = \mathbf{J} = \mathbf{A}$. In soft-body motion, as there are no restrictions on the motion, no global transformation can be applied on the normal vectors, as in the rigid-body motion case.

As in many computer graphics disciplines there is always discrepancy between fast computation and exact computation. Some time critical (real-time) applications require fast computation and some inaccuracies up to some level are excused, a typical example is shading. Other applications require the highest possible precision and may wait for an exact result if necessary, e.g., point containment tests are very critical and some inaccuracy in the result (inside/outside) may have large consequences.

In this paper we will deal with face normal computation of meshes under linear soft-body deformation, i.e., each vertex trav-

---

[*] e-mail: jparus@kiv.zcu.cz
[†] e-mail: aht@hig.se
[‡] e-mail: kolinger@kiv.zcu.cz

Figure 1: A Triangle deforms from the initial position $\mathbf{a_0}, \mathbf{b_0}, \mathbf{c_0}$ to the final position $\mathbf{a_1}, \mathbf{b_1}, \mathbf{c_1}$.

Figure 2: : (a) linear interpolation, (b) quadratic interpolation, (c) higher degree interpolation.

els independently along its linear trajectory. In section 2 we will describe five approaches for face normal interpolation. In section 3 we will compare described approaches from the time consumption and accuracy point of view and we will discuss pros and cons of each method. Section 4 concludes the paper together with some ideas for the future work.

2 Face Normal Interpolation

First let us describe more specifically our setting. We suppose that mesh is deforming in time so that only the position of vertices varies; the connectivity (i.e., interconnection of vertices by edges) is always the same. We know the current state of the mesh at time t_0 and we also know the future state of the mesh at time t_1, which is usually given by some simulation process (e.g., collision detection). An individual state of the mesh will be referred to as a keyframe. We suppose that the trajectory of vertices between individual keyframes is linear. More complicated curved trajectories can be approximated by piecewise linear curves and our approaches can be applied separately for individual linear segments of the trajectory. As the triangular mesh is just a set of triangles we will first consider only one moving tri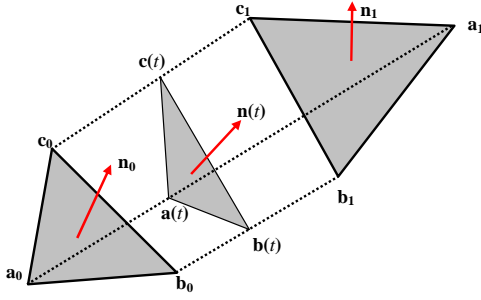angular face. The setting is demonstrated in figure 1. A triangle deforms from the initial position $\mathbf{a_0}, \mathbf{b_0}, \mathbf{c_0}$ to the final position $\mathbf{a_1}, \mathbf{b_1}, \mathbf{c_1}$. The vertex trajectories $\mathbf{a}(t), \mathbf{b}(t), \mathbf{c}(t)$ are linear. We can easily compute the initial normal $\mathbf{n_0}$ (for the initial position of the face) and the final normal $\mathbf{n_1}$ (for the final position of the face). We are interested in how to compute the face normal $\mathbf{n}(t)$ when the face deforms between the initial position to the final position so that it more or less respects the triangle deformation.

The simplest approach is to recompute the face normal in each time instant by taking the cross-product of the edges, i.e.:

$$\mathbf{n} = (\mathbf{v_1} - \mathbf{v_0}) \times (\mathbf{v_2} - \mathbf{v_0}) \qquad (1)$$

where $\mathbf{v_0}, \mathbf{v_1}, \mathbf{v_2}$ are vertices of triangle. The result is exact, but it requires a cross-product computation (9 additions, 6 multiplications) and normalization and therefore we are looking for other methods, which are perhaps less accurate but faster. Anyhow, they need to approximate the normal in a good way.

2.1 t-variant cross product

We can use the usual cross-product approach but instead of using vertex coordinates we use the trajectories of vertices [Parus 2006], i.e.:

$$\mathbf{n}(t) = (\mathbf{b}(t) - \mathbf{a}(t)) \times (\mathbf{c}(t) - \mathbf{a}(t)) \qquad (2)$$

Since the vertex trajectories are linear then the result is a degree two polynomial for each component of the normal. If there are two
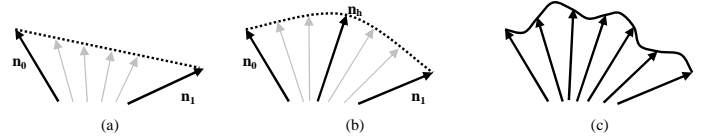
trajectories with the same direction and the same length then equation (2) is degree one polynomial. If all three trajectories have the same direction and the same length then equation (2) is constant, i.e., not time dependent. By evaluating the degree two polynomials we obtain the exact normals. The advantage of this approach is that the coefficients of the degree two polynomials can be precomputed in a preprocessing stage. Moreover, the quadratic polynomials can be evaluated by the Horner scheme (6 additions, 6 multiplications). It should be noted that the computation can be reduced further as explained later. The length of the normal is proportional to the area of the face and therefore normalization is required. The normalization can also be included into a t-variant formula:

$$\mathbf{n}(t) = \frac{(\mathbf{b}(t) - \mathbf{a}(t)) \times (\mathbf{c}(t) - \mathbf{a}(t))}{\|(\mathbf{b}(t) - \mathbf{a}(t)) \times (\mathbf{c}(t) - \mathbf{a}(t))\|} \qquad (3)$$

The resulting evaluation is thus more complicated.

2.2 Lagrange interpolation

Linear interpolation (degree one Lagrange interpolation) of normal vectors is described by the following expression:

$$\mathbf{n}(t) = \mathbf{n_0} + t(\mathbf{n_1} - \mathbf{n_0}) \qquad (4)$$

where $\mathbf{n_0}$ is an initial normal and $\mathbf{n_1}$ is the final normal. This approach is used for example in spatial normal interpolation in Phong shading. In fact, here, the vector is treated as if it was a point. It is fast but not sufficient because the intermediate normals are far from being perpendicular to the triangle; furthermore the intermediate normals need to renormalized figure 2(a). Higher degree Lagrange interpolation fits better the true behavior of the face normals but it is always a tradeoff between a better fit and an oscillation due to the higher degree of the interpolation function. Also, unit length is not preserved. It is demonstrated in figure 2. Figure 2 (a) shows linear normal interpolation, and it can be seen that intermediate normals (gray) do not have unit length. Figure 2 (b) shows degree two Lagrange interpolation which needs an additional intermediate normal $\mathbf{n_h}$ to fit a quadratic interpolation curve. Figure 2 (c) shows higher degree Lagrange interpolation where a number of intermediate normals are required to precompute the interpolation curve, it can be seen that due to the high degree of the interpolation polynomial the normal is oscillating.

2.3 Vector SLERP

SLERP (Spherical Linear intERPolation) [Glassner 1999] is a technique for interpolation of vectors, which maintains unit length, it is defined as:

$$SLERP(\mathbf{n_0}, \mathbf{n_1}, t) = \frac{\sin((1-t)\theta)\mathbf{n_0} + \sin(t\theta)\mathbf{n_1}}{\sin(\theta)} \qquad (5)$$

where $\mathbf{n_0}, \mathbf{n_1}$ is the initial and target normal respectively, θ is the angle between $\mathbf{n_0}, \mathbf{n_1}$. This approach preserves unit length normals but again the direction of the normal is far from being perpendicular to the intermediate triangles.
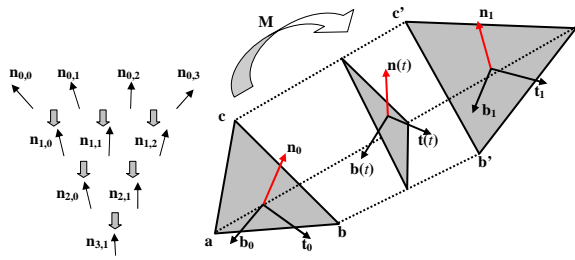
Figure 3: (a) a demonstration of deCasteljau algorithm for vectors, the gray thick arrow represents application of SLERP on two successive normals, (b) quaternion interpolation of face normal, transformation $\mathbf{M}$ transforms the triangle $\mathbf{a}, \mathbf{b}, \mathbf{c}$ to the triangle $\mathbf{a}', \mathbf{b}', \mathbf{c}'$ and $\mathbf{n_0}$ to $\mathbf{n_1}$. The frame $\mathbf{F_1} = (\mathbf{t_1}, \mathbf{b_1}, \mathbf{n_1})$ is computed by transforming $\mathbf{F_0} = (\mathbf{t_0}, \mathbf{b_0}, \mathbf{n_0})$ by the transformation $\mathbf{M}$.

2.4 Spherical deCasteljau

A slightly better idea is to compute several intermediate normals exactly (as in higher degree Lagrange interpolation) and interpolate these vectors on the surface of the unit sphere. In this case, a generalized deCasteljau algorithm for spherical interpolation can be used. The deCasteljau algorithm is well known for fast generation of Bezier curves. It is basically a recursive subdivision of the control polygon of the Bezier curve which converges very quickly to the curve. For example, let us have n vertices of a control polygon and we want to compute the point on the Bezier curve defined by the control polygon for the parameter value t_0. We subdivide each of the $n-1$ edges of the control polygon in the ration $t_0 : (1 - t_0)$. It results in a new control polygon with $n-1$ vertices and $n-2$ edges. The same procedure is recursively repeated until only one edge remains and by subdivision of this edge we end up with a point on the curve. The generalization of deCasteljau algorithm for fast vector interpolation means that we replace line segments with shortest great circle arcs, i.e., the line segment subdivision step is replaced by SLERP of consecutive intermediate normals.

It is demonstrated in figure 3(a) where we have an initial normal $\mathbf{n_{0,0}}$, a final normal $\mathbf{n_{0,3}}$, and two intermediate normals $\mathbf{n_{0,1}}$ and $\mathbf{n_{0,2}}$, (e.g., in the time $t = 0.33$ and $t = 0.66$). When we want to compute normal $\mathbf{n_f}$ at time t we compute normals $\mathbf{n_{1,0}}$, $\mathbf{n_{1,1}}$, $\mathbf{n_{1,2}}$ by applying SLERP on pairs of successive normals, i.e., $\mathbf{n_{1,0}} = SLERP(\mathbf{n_{0,0}}, \mathbf{n_{0,1}}, t), \mathbf{n_{1,1}} = SLERP(\mathbf{n_{0,1}}, \mathbf{n_{0,2}}, t), \mathbf{n_{1,2}} = SLERP(\mathbf{n_{0,2}}, \mathbf{n_{0,3}}, t)$. This process is repeated until one single normal $\mathbf{n_{3,1}}$ is obtained.

2.5 Quaternion SLERP

In this section we will present our idea which is based on the use of quaternions [Shankel 2000] in order to interpolate the face normals. First let us recall two important identities, which we will use in the following description:

Definition 1: Rotation in 3D around an axis $\mathbf{a}$ by an angle ϕ is represented by a 3x3 rotation matrix $\mathbf{R}$ or by a quaternion $\mathbf{q}$ [Shoemake 1985], [Eberly 2004]. Both representations are equivalent.

Definition 2: The matrix $\mathbf{R}$ of a rotation transformation is orthogonal and its columns (and rows) are having unit length, thus the matrix of the rotation forms an orthonormal frame.

The central idea is to set an orthogonal frame $\mathbf{F_0} = (\mathbf{n_0}, \mathbf{t_0}, \mathbf{b_0})$ for the initial face and set an orthogonal frame $\mathbf{F_1} = (\mathbf{n_1}, \mathbf{t_1}, \mathbf{b_1})$ for the final face. To set a frame $\mathbf{F} = (\mathbf{n}, \mathbf{t}, \mathbf{b})$ means to associate one vector of the frame with the normal of the face $\mathbf{n}$, choose a tangent $\mathbf{t}$ which lies in the plane of the face (e.g., an edge of the triangle) and compute the binormal $\mathbf{b}$ by taking the cross product of $\mathbf{n}$ and $\mathbf{t}$, i.e., $\mathbf{b} = \mathbf{n} \times \mathbf{t}$. By organizing the column vectors $\mathbf{t}, \mathbf{b}, \mathbf{n}$ into a 3×3

matrix we obtain a rotation matrix $\mathbf{R}$ which can be converted into a quaternion representation $\mathbf{q}$ [Eberly 2004]. In this way we obtain quaternions $\mathbf{q_0}$ and $\mathbf{q_1}$, representing the initial and the final orientation of the face, respectively. To obtain intermediate normals $\mathbf{n}(t)$, quaternions are interpolated using quaternion SLERP (generalization of vector SLERP for quaternions in 4D), i.e.,

$$\mathbf{q}(t) = SLERP(\mathbf{q_0}, \mathbf{q_1}, t) \tag{6}$$

and intermediate quaternion $\mathbf{q}(t)$ can be converted back to the orthogonal matrix $\mathbf{R}(t)$ and the normal is extracted from the last column of $\mathbf{R}(t)$. It is demonstrated in figure 3(b) where the triangle $\mathbf{a}, \mathbf{b}, \mathbf{c}$ with the frame $\mathbf{F_0}$ is transformed in the triangle $\mathbf{a}', \mathbf{b}', \mathbf{c}'$ with the frame $\mathbf{F_1}$. The intermediate quaternion $\mathbf{q}(t)$ is converted to the frame $(\mathbf{t}(t), \mathbf{b}(t), \mathbf{n}(t))$ and the intermediate normal $\mathbf{n}(t)$ is extracted. The question is how to set the frames $\mathbf{F_0}$ and $\mathbf{F_1}$. We use following scheme. First we compute a transformation matrix $\mathbf{T}$ which transforms vertices of the initial triangle to the vertices of the final triangle, moreover we want that the transformation $\mathbf{T}$ transforms also the initial normal to the final normal. All condition can be expressed by a matrix equation, i.e.:

$$\mathbf{M}[\mathbf{a_0} \mathbf{b_0} \mathbf{c_0} \mathbf{n_0}] = [\mathbf{a_1} \mathbf{b_1} \mathbf{c_1} \mathbf{n_1}] \tag{7}$$

where $\mathbf{a_0}$, $\mathbf{b_0}$, etc. are column vectors, e.g., $\mathbf{a_0} = [\mathbf{a_{0_x}}, \mathbf{a_{0_y}}, \mathbf{a_{0_z}}, 1.0]^{\mathbf{T}}$. The matrix $\mathbf{M}$ can be computed as:

$$\mathbf{M} = [\mathbf{a_1} \mathbf{b_1} \mathbf{c_1} \mathbf{n_1}][\mathbf{a_0} \mathbf{b_0} \mathbf{c_0} \mathbf{n_0}]^{-1} \tag{8}$$

Then we set an arbitrary frame $\mathbf{F_0}$ for the initial face. For the frame $\mathbf{F_1}$ we have to choose $\mathbf{t_1}$ and $\mathbf{b_1}$ since $\mathbf{n_1}$ is given. We compute $\mathbf{b_1}$ by transforming $\mathbf{b_0}$ by the matrix $\mathbf{M}$, i.e., $\mathbf{b_1} = \mathbf{M} \mathbf{b_0}$, $\mathbf{t_1}$ is then computed as cross product of $\mathbf{n_1}$ and $\mathbf{b_1}$, i.e., $\mathbf{t_1} = \mathbf{n_1} \times \mathbf{b_1}$.

3 Comparison and Discussion

We compared the approaches described in section 2 from two points of view. The first aspect is the quality of the interpolation and the second aspect is the time consumption. The quality of the interpolation is measured as follows. The time interval (usually $[0, 1]$) is sampled and in each sample i an angle a_i between the exact normal (computed by the cross product) and interpolated normal (computed by some interpolation approach described in section 2) is computed. Angles between the exact normals and interpolated normals in individual samples are summed and the resulting number q represents the quality of the interpolation scheme, i.e. $q = \sum a_i, i = 1...n$ where n is the number of samples.
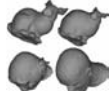
Normal interpolation approach	# faces: 15300	# faces: 44700	# faces: 35180	# faces: 21040
Linear	18916	55756	44078	20061
Quadratic	4820	15792	9681	5425
Cubic	1906	6444	3640	1572
t-variant cross product	0	0	0	0
Vector SLERP	19427	57999	44972	20861
Spherical deCasteljau	7039	21772	16068	7647
Quaternion SLERP	17766	53492	42831	19202

Table 1: Comparison of different normal interpolation approaches.

We tested different interpolation approaches on a morphing animation, where a mesh composed of triangles deforms from one shape to another shape so that the trajectories of individual vertices are linear. The numbers in table 1 represents a quality of interpolation for the whole mesh, it is computed as a sum of quality of interpolation of individual triangles of the mesh, i.e., $Q = \sum q_j, j = 0...m$

where m is the number of triangles of the mesh and q_j is the quality of interpolation of j-th triangle. The numbers in table 1 must always be viewed with respect to the number of triangles and the number of samples. Rather then absolute values it is more important to compare ratios between different methods, e.g., it can be seen that quadratic interpolation is almost 4-times better than simple linear interpolation. The second, third and fourth row shows the results of normal interpolation described in Section 2.2. The row t-variant cross product shows the result of the approach from the Section 2.1. The row Vector SLERP shows results of method described in the Section 2.3. Results of vector interpolation using generalized deCasteljau algorithm (Section 2.4) are shown in the Spherical deCasteljau row. The last row shows the quality of normal interpolation using quaternions.

From table 1 it is clear that the best approach from the quality point of view is the t-variant cross product approach which gives exact normals, i.e., it is not really an interpolation approach. The other approaches (except the quaternion approach) handles normal vectors as usual vectors, i.e., it is not respected that normals vector are perpendicular to some surface and in fact, these approaches can be used for the interpolation of any vectors. The second worst result is obtained by linear interpolation. Better results can be achieved by quadratic or cubic interpolation (up to 90% improvement). The worst result was achieved by SLERP of normal vectors. The quality of interpolation by spherical deCestaljau approach depends on how many intermediate normals we use. In this case we used initial and final normal and two additional intermediate normals. The quaternion SLERP approach has slightly better results (on average about 5%) than simple linear interpolation.

The problem of Quaternion SLERP approach is that there are an infinite number of frames configurations that represent the face orientation. First, we generated a pair of random frames $\mathbf{F_0}$ (for the initial face), $\mathbf{F_1}$ (for the final face) and we tested the quality of interpolation. A random frame is given by a normal $\mathbf{n}$ of the face, an arbitrary vector $\mathbf{t}$ lying in the plane of the face and a binormal vector $\mathbf{b}$ computed as $\mathbf{b} = \mathbf{n} \times \mathbf{t}$. Different configurations of random frames lead to different quality of interpolation. We did not succeed in devising a deterministic algorithm for computation of best pair of frames configuration (from the interpolation quality point of view). Our frame computation based on the transformation matrix in equation (8) which maps the initial face to the final face yields good results, however some specific frame configurations led to a better interpolation quality.

Next we will compare various approaches from the time consumption point of view. We will not present exact timing since it is dependent on how various elementary operations (SLERP, cross-product, polynomial evaluation, etc.) are implemented. Some of them can be implemented in hardware so that their execution can be very fast. We will express the time consumption in terms of elementary operations, so that one must always decide which approach is the most suitable according to the actual application platform.

Elementary operations used in table 2 are polynomial evaluation, SLERP and quaternion to matrix conversion. Polynomial evaluation is used in the Lagrange interpolation approach and in the t-variant cross product approach. Polynomials can be evaluated by the Horner scheme which saves some multiplications in comparison with usual evaluation of polynomial in monomial form. SLERP is used in the Vector SLERP approach and in the deCasteljau approach. SLERP requires evaluation of trigonometric functions which are computationally expensive, but it can be speed up by an incremental approach which was described Barrera et al in [Barrera 2004].

Quaternion to matrix conversion is used in the Quaternion SLERP approach. Note that in our case we need to extract only one column from the matrix, i.e., the normal. It is also important to decide whether we need random access to the deformation or just

Method	Requires normalization	Computation
Linear interpolation	yes	3 linear interpolation, Eq. (4)
Quadratic, cubic interpolation	yes	3 evaluation of degree two (quadratic interp.) or degree three (cubic interp.) polynomials
t-variant cross product	yes (Eq. 1) no (Eq. 2)	3 evaluation of degree two polynomials (Eq. 2) 3 evaluation of rational polynomial (Eq. 3)
Vector SLERP	no	1 SLERP
Spherical deCasteljau	no	n(n-1)/2 SLERPs, where n is the number in precomputed normals
Quaternion SLERP	no	1 SLERP, quaternion to matrix conversion

Table 2: Comparison of various normal interpolation approaches from time consumption point of view in terms of elementary operations.

sequential access. Random access means that we can jump from one time instant to another. Sequential access means that the mesh deformation is sequentially played; in this case incremental methods (using temporal coherence) for computing SLERP [Barrera 2004] or polynomial evaluation [Hast 2003] can be used. For example, a degree two polynomials can be evaluated by an incremental method using only two additions.

4 Conclusions and Future Work

In this paper we showed five approaches on how to compute normals of triangular faces under linear deformation. It is clear that the Linear interpolation approach always will be faster than any other approach but it will also have very bad results when the deformation is dramatic. The higher degree Lagrange interpolation is better but still it is just an interpolation of vectors and in some cases it does not capture the face deformation well.

The t-variant cross product approach gives exact face normals; the lengths of normals are proportional to the size of face so it can be directly used for vertex normal computation weighted by the area [Parus 2006]. If unit length normals are required, both Lagrange interpolation and t-variant cross product approaches require additional normalization.

Approaches based on SLERP do not require vector normalization since the vectors are interpolated on the surface of the unit sphere. The problem of Spherical deCasteljau approach is that it interpolates the initial and final normal but it just approximates the intermediate normals (in the same way how Bezier curve approximates the control polygon). So it is not useful when we want interpolate exactly some vector different then the initial and the final.

The new quaternion interpolation approach is different from previous approaches (except t-variant cross product approach) because it interpolates frames instead of vectors. Interpolation of the frame may capture better the true triangle motion and thus it can produce more accurate normals then the simple Vector SLERP approach. Like the linear interpolation approach and the Vector SLERP approach it has the advantage that no intermediate normals are necessary to compute exactly in order to set up the interpolation for the animation.

In the future work we would like to study the Quaternion interpolation approach. It has promising results because by experiments we found out that some frame configurations lead to a better quality of interpolation than others, so we would like to find such frame configurations which lead to the best possible quality of the normal interpolation.

References

T. Barrera, A. Hast, E. Bengtsson 2005. *Incremental Spherical Linear Interpolation* SIGRAD 2004, pp. 7-10.

D. EBERLY 2004. *Game Physics* Morgan Kaufman, 2004

A. GLASSNER 1999. *Situation Normal* Andrew Glassner's Notebook- Recreational Computer Graphics, Morgan Kaufmann Publishers, pp. 87-97.

J. GOMES 1999. *Warping and Morphing of Graphical Objects* Morgan Kaufman. San Francisco, California. 1999.

H. GOURAUD 1971. *Continuous Shading of Curved Surfaces* IEEE Transactions on Computers, Vol. 20, No. 6, 1971.

A. HAST, T. BARRERA, E. BENGTSSON 2003. *Improved Shading Performance by avoiding Vector Normalization,* WSCG'01, Short Paper,2001, pp. 1-8.

S. JIN 2005. *A Comparison of Algorithms for Vertex Normal Computation* Communications of the ACM, Vol. 18, No 6, 1975

Z. KARNI 2004. *Compression of soft-body animation sequences* Computers and Graphics, 28: 25-34. 2004.

J. PARUS, I. KOLINGEROVÁ 2006. *Normal Evaluation for Soft-body Deforming Meshes* SimVis2006, pp. 157-168, 2006.

B. T. PHONG 1975. *Illumination for Computer Generated Pictures* The Visual Computer, Springer-Verlag GmbH, Issue: Vol. 21, No 1-2, pp. 71-82, Feb. 2005.

J. SHANKEL 2000. *Interpolating Quaternions* Game Programming Gems. Edited by M. DeLoura. Charles River Media, pp. 205-213

K. SHOEMAKE 1985. *Animating rotation with quaternion curves* ACM SIGGRAPH, pp. 245-254.

A multi-sampling approach
for smoke behavior in real-time graphics

Henrik Gustavsson[†]
University of Skövde

Henrik Engström[‡]
University of Skövde

Mikael Gustavsson[*]

Abstract

Smoke simulation is a key feature of serious gaming applications for fire-fighting professionals. A perfect visual appearance is not of paramount importance, the behavior of the smoke must however closely resemble its natural counterpart for successful adoption of the application. We therefore suggest a hybrid grid/particle based architecture for smoke simulation that uses a cheap multi-sampling technique for controlling smoke behavior. This approach is simple enough for it to be implemented in current generation game engines, and uses techniques that are very suitable for GPU implementation, thus enabling the use of hardware acceleration for the smoke simulation.

Keywords: Smoke Simulation, Particle System, Multi-sampling, GPU, Serious Gaming

1 Introduction

Smoke simulation is a topic that has received much attention from computer graphics researchers. Commonly, work on smoke simulation is focused on graphical appearance, either producing photo realistic smoke [Fedkiw et al. 2001, Losasso et al. 2004] or less photo-realistic, cartoon-like smoke [Selle et al. 2000]. When simulated smoke is used in animations, graphical appearance is considered much more important than real-time execution of the particle system. In a serious gaming application, the behavior of the smoke is more important than the appearance. Computational efficiency is also important since a computer games engine is used for rendering as well as other tasks related to the game. For realistic behavior and appearance of smoke, rendering of a single frame of animated smoke typically takes up to a minute [Losasso et al. 2004] even very old, and comparably simplistic solutions [Ebert and Parent 1990] are prohibitively expensive to implement in a current generation game engine.

Particle systems in computer games typically use simple rule-based engines executed by the CPU of the computer used to run the game engine [Sele et al. 2000]. Such an approach tends to produce very simple particle systems with simplistic behavior and very limited interaction with the environment. We suggest an approach which uses a simple multi-sampling technique for a particle system that is similar to a rule-based particle engine but has a behavior that is more similar to advanced particle systems. This engine is sufficiently simple that it can be implemented in a current generation game engine, and in the future implemented using a GPU for hardware acceleration of the computations.

[†] e-mail: henrik.gustavsson@his.se

[‡] e-mail: henrik.engstrom@his.se

[*] e-mail: mikgust@gmail.com

2 Related Work

Particle systems are available in all of the major 3d animation packages and game engines. The particle engines available in computer games are however much less sophisticated than their counterparts in 3d animation software. This is mainly due to the stringent real-time requirements of computer games. Recently, several authors have investigated hardware acceleration of particle systems by offloading particle calculations to the GPU [Kipfer et al. 2004]. This would allow simulation of a significantly larger set of particles on the same hardware. If main memory can be avoided by using the GPU, the strain on the often congested system bus can be reduced significantly. There have also been suggestions for executing more advanced particle engines on the GPU such as fluid simulation [Amada et al. 2004, Hegeman et al. 2006]. Even with GPU acceleration, these systems are not sufficiently fast to simulate a particle system in a game engine.

Serious gaming is a topic that has recently emerged as a light-weight tool for training professionals or military personnel. Existing video game engines can be readily adapted for training [Fong 2004]. To the player, a video game consists of a playing environment, characters, tools, and missions. These elements can then be altered to better fit for training purposes. There are several games available today for various types of training. Many of these are military simulators, but others have been developed, such as games for training of firefighting professionals [Stapleton 2004]. A key issue when it comes to the use of games for training of professionals is user acceptance. Acceptance is related to the perceived realism of the simulated environment and poor acceptance may lead to reduced efficiency of the training [Lampton et al. 2002]. In a game for fire-fighting professionals such elements as the color of clothing and details of equipment may be important for user acceptance [Stapleton 2004]. In a game focused on extinguishing fires, the behavior of the smoke is one of the most important properties for user acceptance. If the smoke does not behave realistically the game is less useful for training purposes.

Currently, there are environments such as NIST Fire Dynamics Simulator (FDS) available for fire simulation in virtual buildings [Hyeong-Jin and Lilley 2006] or even smaller structures [Hamis 2006] which portray fire behavior in a very realistic manner. These simulations are very realistic but the performance is not high enough to be suitable for real-time execution using a game engine. There has also been work in producing realistic graphical representation of fire [Nguyen et al. 2002], but since fire simulation uses many of the same techniques as in smoke simulation fire simulation it has most of the same drawbacks as smoke simulation for real-time games environments.

3 A simplified physics model

The most common approach to realistic smoke motion is to either use Navier-Stokes or Euler equations. For a standard simulation system, the equation might use variables such as density, pressure, volume force, velocity and viscous coefficients [Fedkiw 2001]. The equations are then used to compute the properties of the fluid or smoke for each element of a grid. In a Lagrangian or grid-less approach, particles are often updates using a variations of the Navier-Stokes equations used by the grid methods where the acceleration of the particle is decided by the velocity, density and additional forces that may affect the particle.

For this work a Lagrangian approach was selected since particles can be updated using a simplified equation based on the pressure surrounding the particle. In the simplified context of smoke simulation for an in-door fire, the equations that affect particle motion can be simplified significantly. Firstly, there are no external forces such as wind inside a building. Furthermore, since the smoke is light, the effect of gravity is also negligible. There are thus two main forces that affect the smoke, the pressure of surrounding particles and the upward motion due to the heat generated from the fire.

We suggest an approach that uses a medium sized 3D grid implemented as a 3D texture that contains the density of particles in each of the grid cells. Each particle is controlled by the pressure in the surrounding grid elements, which is derived through texture sampling. The size of the grid needs to be roughly the size of the polygons used to render the smoke so that the polygons cluster together to form a solid block when the maximum density has been reached. A low density grid cell should attract particles and a high density grid cell should repel particles. A function can then be used to control the strength of the effect on the particle dependent on the pressure in grid cell. This gives us the equation:

$$\mathbf{a}_i = \frac{\sum \mathbf{s}_j \cdot f(\rho_i)}{\sum f(\rho_i)}$$

Figure 1. Particle acceleration equation

Where $\mathbf{a}$ is the acceleration for particle i, $\mathbf{s}$ is the sample vector and f the attraction function and ρ is the particle density at the position of the particle i offset by the sample vector $\mathbf{s}$. We simplify this further by using the acceleration as the particle velocity, thereby removing the need to keep track of both the particle position, velocity and acceleration.

This very simple approach also allows simplification of the particle updates since particle to particle collisions are not required and collisions with the environment can be simulated by grid elements with maximum density. This means that the particle update code will only need to know the density of particles and the particle position. A standard particle system commonly uses the position, velocity and acceleration of the particles [Kipfer et al. 2004]. By using the acceleration as the velocity, the update procedure is even simpler than a standard particle system.
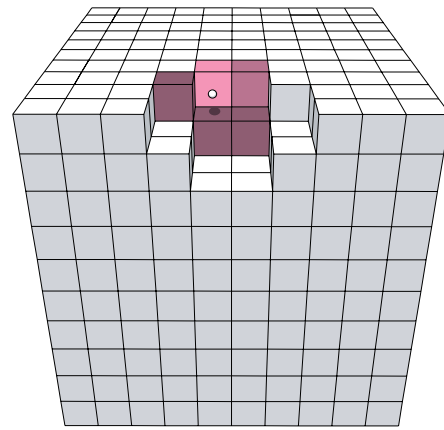

Figure 2. Multi-sampling hybrid grid/particle approach

4 Multisampling approach

When multisampling is used, it is important to select a sampling strategy that gives the best output for the least amount of samples. In the case of the particle engine it is important that the sampling occurs farther away from the location of the particle than the closest grid so that the particle will be attracted to low density areas some distance away from its current location. For this application we chose a sampling pattern that samples all 26 grid points one unit away from the particle so that no low density grid is missed regardless of the direction. The second level of samples are taken two steps away from the centre grid in a pattern similar to the Nvidia quincunx (pattern looks like the number 5 on a die) pattern used for anti-aliasing. The second level sample vectors are twice as long as the first level but since there are fewer of them, the total influence of the second level is very similar to the first level. On the third level there are only sample vectors for the six ordinal directions. so even though the vectors are longer, total influence is lower than both the first and second level.
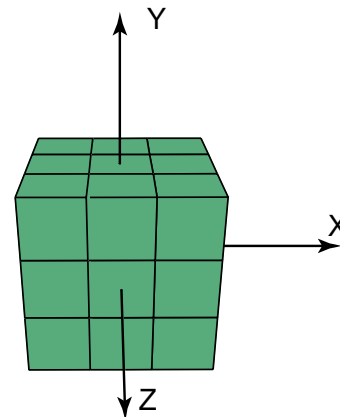

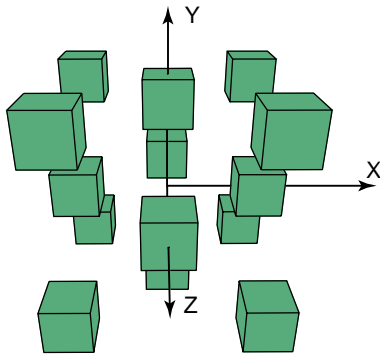**Figure 3. First level (26 samples)**
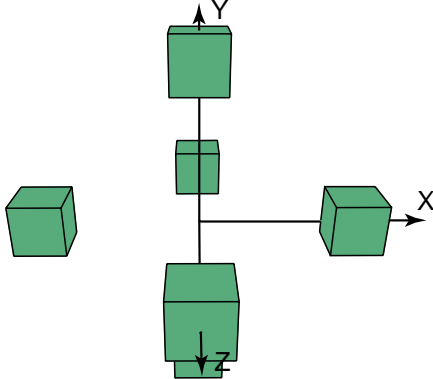
Figure 4. Second level (14 samples)

Figure 5. Third level (6 samples)

When the sampling pattern has been decided it is necessary to select the function that will decide how the particles will react. A high value means that the particles will be attracted and a value below zero means that the particles will be repelled. The steepness of the curve determines how fast that will happen and how many particles that will end up in each element of the grid.

For this application we chose a variation of the logistics function, with parameters that gave us a sigmoid curve that has the value 1 at 0.0 and reaches zero at 5 and -1 at 10. This means that it is unlikely that a grid element will contain more than 5 particles since any value above 5 will repel particles. For a room that has a very large number of grid elements it is necessary to use a steeper curve
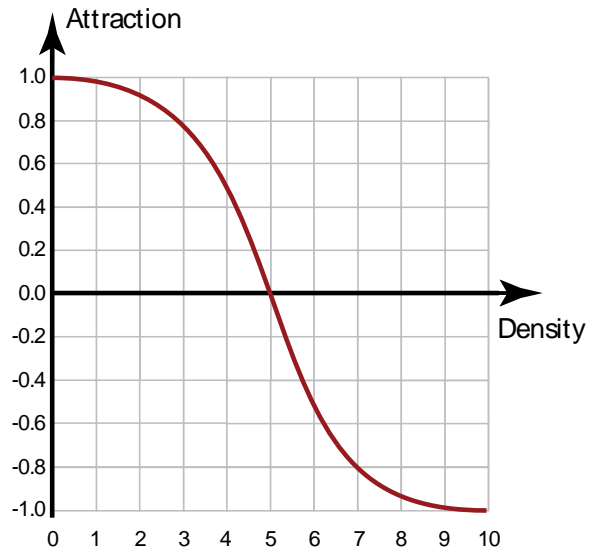

Figure 6. Sigmoid particle density function

If a simulation is started without any bias toward a specific direction, there is no incentive for the smoke particles to rise since all directions have equal influence on the particle speed. To introduce a system where the particle can rise while it is hot and close to the fire we must introduce a bias for each of the sample points so that when the particle is hot the sample points above 0 on the y axis get a slight advantage. This advantage can be reduced, either when the particle becomes older or when the particle has been in a low density (fewer warm particles) environment for a number of time steps. These parameters can be recorded into a bias texture, an example of which is shown below. Note that the differences between high and low bias has been exaggerated.
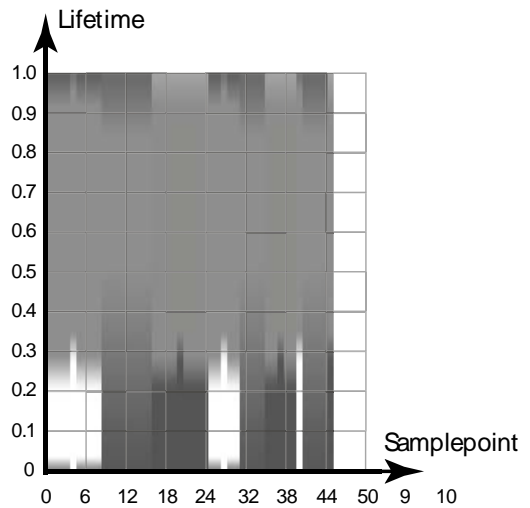

Figure 7. Sigmoid particle density function

The resulting pseudo-code can be seen below. Note that three textures are used, one 3D texture for the grid, one 2D texture for the bias and one 1D texture for the sigmoid function lookup table. This means that each particle will require 138 samples per particle which is not much since a GPU can perform millions if not billions of samples per second.

```
for(int j=0;j<46;j++){
        x=xk+sample_x[j];
        y=yk+sample_y[j];
        z=zk+sample_z[j];

        int griddata=Grid[x][y][z];
        cnt+=abs(sigmoid[griddata]);

        sumx+=sigmoid[griddata]*
        sample_x[j]*bias[lifetime][idx];
        sumy+=sigmoid[griddata]*
        sample_y[j]*bias[lifetime][idx];
        sumz+=sigmoid[griddata]*
        sample_z[j]*bias[lifetime][idx];
}
```

Figure 8. Particle update pseudo-code

5 Results

Through visualization of the density grid it is possible to see that the particle system works well when it comes to equalizing the particle pressure. The particles strive to move away from the high density areas into lower density areas. The largest dot at the top of figure 8 repels surrounding particles with maximum force.
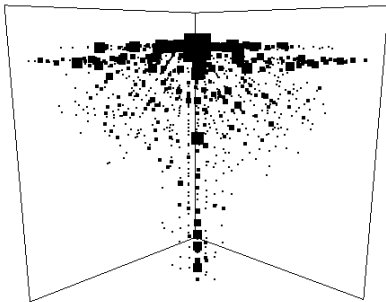

Figure 9. Particle density plot

Particle behavior over time is also close to the expected result, the hot recently born particles rise with a high speed and the colder older particles are more stationary or are even dropping slightly when cold. The figures below depict the simulation at t=1 second, t=10 seconds and t=25 seconds and finally t=50 seconds.
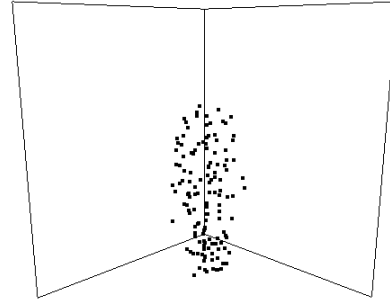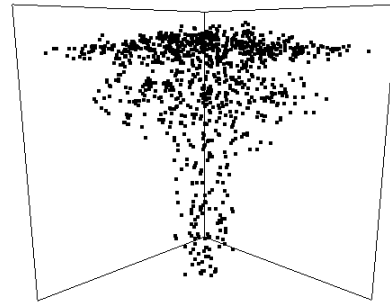

Figure 10. Simulation at t=1 second
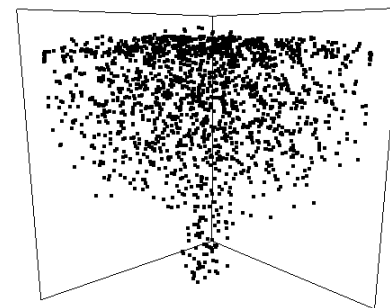

Figure 11. Simulation at t=10 seconds

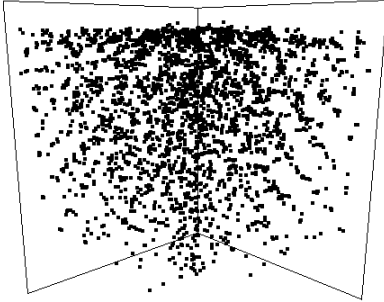**Figure 12. Simulation at t=25 seconds**

20

Figure 13. Simulation at t=50 seconds

6 Conclusions

We have shown that it is possible to construct a very simple particle system that produces smoke that behaves realistically in that the smoke goes up to the top of the room and then slowly fills the room from above until the room is filled with a uniform cloud of smoke. The simulation uses only approximately 150 texture samples per particle which means that thousands of particles can be simulated on a modern computer without large negative effects on the performance of the game engine.

The bias texture has a big effect on the behavior of the particles and construction of the bias texture does require some attention for realistic behavior. Similarly, the particle emission and density also affects the visual quality. If the grid size is too large or particles are released at locations that are very close, particles will not move realistically.

7 Future Work

There are large amounts of future work to conduct with this smoke simulator.

- Examining the effects of the bias texture and the sigmoid function and see if other values or configurations are better than the current.

- More realistic rendering through billboards for smoke-puffs and using the density grid as a basis for putting soot on the walls of the room.

- Benchmarking the solution in real-life situations to see where for instance the number of particles gives the best visuals and the lowest negative effect on game performance

- One of the most pressing issues is the implementation of this smoke simulator in a real games engine to see how it reacts in a real game. Another pressing issue is to make the GPU implementation and examine the effect of the GPU on for example number of particles.

8 References

AMADA, T., IMURA, M., YASUMURO, Y., MANABE, Y. AND CHIHARA, K. 2004. *Particle-Based Fluid Simulation on GPU*, ACM Workshop on General-Purpose Computing on Graphics Processors and SIGGRAPH 2004 Poster Session.

EBERT, D. S. AND PARENT, R. E. 1990. *Rendering and Animation of Gaseous Phenomena by Combining Fast Volume and Scanline A-buffer Techniques.* Computer Graphics (SIGGRAPH 90 Conference Proceedings) 24(4):357–366

FEDKIW, R., STAM, J. AND WANN JENSEN, H. 2001. *Visual Simulation of Smoke*. ACM SIGGRAPH 2001, 23-31.

FONG, G. 2004. *Adapting COTS games for military simulation.* Proceedings of the 2004 ACM SIGGRAPH International Conference on Virtual Reality Continuum and its Applications in Industry, 269-272.

HEGEMAN, K., CARR, N. A. AND MILLER, G. S. P. 2006. *Particle-Based Fluid Simulation on the GPU*, International Conference on Computational Science 2006, Part IV, LNCS 3994, 228–235

HAMINS, A., BUNDY, M. AND DILLON, S. E. 2006. *Characterization of Candle Flames.* 44th AIAA Aerospace Sciences Meeting and Exhibit. 1-13

HYEONG-JIN, K. AND LILLEY, D. G. 2006. *Computer Modeling of Developing Structural Fires.* 44th AIAA Aerospace Sciences Meeting and Exhibit

KIPFER, P., SEGAL, M. AND WESTERMANN, R. 2004. *UberFlow: A GPU-Based Particle Engine.* HWWS'04 Proceedings of the ACM SIGGRAPH / EUROGRAPH conference on graphics hardware.

LAMPTON, D. R., BLISS, J. P. AND MORRIS, C. S. 2002. *Human performance measurement in virtual environments.* K. M. Stanney (Ed.), Handbook of virtual environmentvs: Design, implementation, and applications, Mahwah, NJ: Lawrence Erlbaum Associates, 701-720

LOSASSO, F., GIBOU, F. AND FEDKIW, R. 2004. *Simulating water and smoke with an octree data structure.* ACM Transactions on Graphics (TOG) 23(3) 57–462

NGUYEN, D. Q., FEDKIW, R. AND WANN JENSEN, H. 2002. *Physically based modeling and animation of fire.* Proceedings of the 29th annual conference on Computer graphics and interactive techniques. 721-728

STAPLETON, A. 2004. *Serious Games: Serious Opportunities.* Paper presented at the Australian Game Developers Conference, Academic Summit, Melbourne

SELLE, A, MOHR, A. AND CHENNEY, S. 2000. *Cartoon Rendering of Smoke Animations.* ACM SIGGRAPH

Interactive Simulation of Elastic Deformable Materials

Martin Servin*
Department of Physics,
Umeå University

Claude Lacoursière†
HPC2N/VRlab and
Computing Science Department,
Umeå University

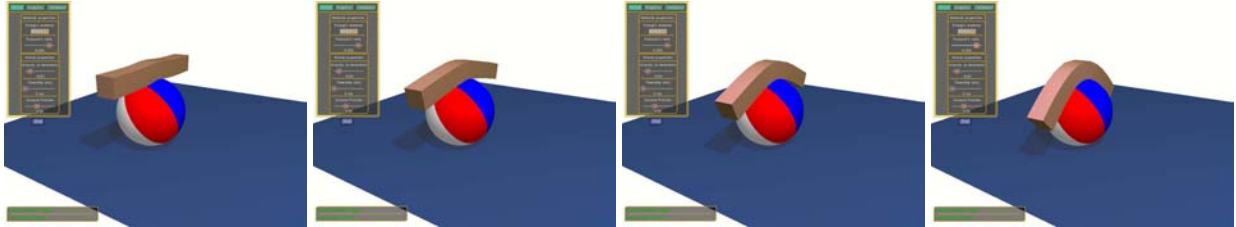Niklas Melin
Department of Physics,
Umeå University

Figure 1: Snapshot from interactive simulation of a deformable beam and solid sphere

Abstract

A novel, fast, stable, physics-based numerical method for interactive simulation of elastically deformable objects is presented. Starting from elasticity theory, the deformation energy is modeled in terms of the positions of point masses using the linear shape functions of finite element analysis, providing for an exact correspondence between the known physical properties of deformable bodies such as Young's modulus, and the simulation parameter. By treating the infinitely stiff case as a kinematic constraint on a system of point particles and using a regularization technique, a stable first order stepping algorithm is constructed which allows the simulation of materials over the entire range of stiffness values, including incompressibility. The main cost of this method is the solution of a linear system of equations which is large but sparse. Commonly available sparse matrix packages can process this problem with linear complexity in the number of elements for many cases. This method is contrasted with other well-known point mass models of deformable solids which rely on penalty forces constructed from simple local geometric quantities, e.g., spring-and-damper models. For these, the mapping between the simulation parameters and the physical observables is not well defined and they are either strongly limited to the low stiffness case when using explicit integration methods, or produce grossly inaccurate results when using simple linearly implicit method. Validation and timing tests on the new method show that it produces very good physical behavior at a moderate computational cost, and it is usable in the context of real-time interactive simulations.

CR Categories: I.3.5 [Computer Graphics]: Physically based modeling— [I.3.7]: Computer Graphics—Virtual reality

Keywords: deformable simulation, elasticity, constrained dynamics, stability, numerical integration

1 Introduction

Simulation of elastically deformable objects is a necessity for creating certain types of virtual environments, such as those designed for surgical or heavy vehicle operator training applications, for instance. This problem has generated interest recently and a survey of well-known methods can be found in [Nealen et al. 2005]. We focus here on numerically stable methods for integrating elastic deformable objects at interactive rates over a wide range of stiffness, including incompressibility and the nearly rigid limit, and on establishing a clear correspondence between the known material parameters used in elasticity theory–such as Young's modulus–to the simulation parameters.

We will briefly describe the most commonly used techniques and explain how they fail for high stiffness, either losing stability, yielding grossly inaccurate results, or both, and how they lack a clear correspondence between simulation parameters and physical properties. Observe that both full incompressibility as well as full rigidity are cases of infinite stiffness. We argue that the key to handling the *infinite stiffness* regime is to reformulate the problem as a kinematically constrained dynamical system. The recovery of *finite stiffness* is done using a constraint regularization and stabilization technique which amounts to a numerically stable penalty technique with the strain energy as the penalty function. Formulation of constrained mechanical systems and techniques for solving them have been presented before in the graphics literature in [Witkin et al. 1990] [Baraff 1996] and [Erleben et al. 2005]. The idea of using standard energy terms for generating penalty force has also been used several times before [Terzopoulos et al. 1987][Teschner et al. 2004]. It is the mixing of these two ideas in a numerically stable way which is novel, and this is achieved by constructing a special integrator which is semi-implicit [Lacoursière 2006] in combination with appropriate formulation of energy and dissipation terms. The resulting method can handle stiffness from zero to infinity without developing instabilities though the effective stiffness and accu-

racy is limited by the size of the time step. Since stability hinges on reasonably accurate solutions of a large system of linear equations, performance is achieved by exploiting sparsity in a direct solver as was done in [Baraff 1996], for instance. Validation tests are conclusive in demonstrating good agreement between the simulated physical properties and the input parameters. Using a widely available sparse matrix package, UMFPACK [UMFPACK], we achieved linear complexity as a function of system size and sufficiently fast execution for interactive rates for moderately sized systems.

1.1 Contribution and organisation of this paper

Section 2 provides background of particle based simulation of deformable materials. Previous work is reviewed and the advantages and shortcomings of the different strategies are discussed. In section section 3 we consider a simple example that elucidates some of the ideas and pitfalls. We argue that the key to fast and stable simulation even in the stiff regime is to reformulate the problem as a kinematically constrained dynamical system. The new method for simulating elastic deformable materials is presented in section 4. This method combines the technique of regularization and stable stepping of constrained systems – described by Eqs. (20) – and elasticity modeling based on well established material models – specified by Eq. (15). The model is evaluated, through numerical experiments, and discussed in section 5. Summary and conclusions are found in section 6.

2 Particle system models for deformable objects

There are several ways to represent deformable objects but we restrict our attention to those represented as a set of interacting point masses, namely, lumped element models. Elastic properties of the bodies are constructed by defining various forces and constraints on the particles so that all constraints are satisfied and all internal forces cancel out when the body is in the reference configuration. A recent survey of techniques for simulating deformable objects is found in [Nealen et al. 2005]

We denote the particle positions by $x = (\mathbf{x}^{(1)T}, \mathbf{x}^{(2)T}, \ldots, \mathbf{x}^{(N)T})^T$, where $\mathbf{x}^{(i)} = (x_1^{(i)}, x_2^{(i)}, x_3^{(i)})^T$ is the 3D position vector of particle i, with the parentheses emphasizing that it is a particle index rather than a component of a vector, and N is the total number of particles. The particle velocities are $v = \dot{x}$ and the particle masses $m^{(i)}$ are collected in the diagonal mass matrix M of dimension $3N \times 3N$. The equations of motion are:

$$\dot{x} = v \qquad (1)$$
$$M\dot{v} = F_{ext} + F_{int} + F_c \qquad (2)$$

where the total force is divided into external force F_{ext}, internal force F_{int} and constraint force F_c. Sometimes we agglomerate the external and internal forces into $F = F_{ext} + F_{int}$.

2.1 Geometry, energy and force

General penalty forces derived from energy functions for simulating deformable elastic objects were introduced in the graphics literature by [Terzopoulos et al. 1987]. The idea is to define *geometric displacement* functions $\phi_i(x)$, which vanish in the rest configuration, and a potential energy $U(x) = \sum_i (k_i/2)\phi_i^2(x)$, where the k_i's

are positive stiffness parameters. The functions $\phi_i(x)$ might be the local curvature for a string or that of the surface for a membrane instance. Other examples include deviation from rest distance between two particles, i, j, with $\phi_k(x) = (|\mathbf{x}_{(i)} - \mathbf{x}_{(j)}| - L_{ij})/L_{ij}$, where L_{ij} is the rest distance. The resulting energy represents that of a linear spring. The force generated from the potential $U(x)$ is well known to be: $F_{int} = -\partial U/\partial x$. Note that this is linear in $\phi_k(x)$ and therefore, all forces vanish at equilibrium, as needed. Provided the Jacobian matrix $J = \partial\phi/\partial x$ has full rank at the equilibrium point, the restoration forces are *linear* in the geometric displacements.

Therefore, the art of this formulation is to chose displacement functions $\phi_k(x)$ which do not have zero derivatives near the equilibrium, and which are linear independent of each other at or near the equilibrium point. In computing the bending energy for a cable made of point masses for instance, there is a zero derivative condition if one takes the bending displacement variable as: $\phi_k(x) = (|\mathbf{x}_{(i+1)} - \mathbf{x}_{(i-1)}| - 2L_0)/(2L_0)$. There are several such cases which are not so obvious and which arise accidentally when trying to brace a set of particles with springs so as to produce a unique rest configuration that is stable under arbitrary deformations. This sort of problem is usually solved using inverse trigonometric functions, as is done in the construction of shear restoration forces in cloth simulation [Baraff and Witkin 1998]. This can be quite expensive computationally and produce unexpected results.

This energy based penalty formulation was developed systematically for handling elastic solids in [Teschner et al. 2004] by defining three types of *geometric* displacements, namely, *distances*, *volumes*, and *surface areas*. These displacement functions involve the coordinates of two or more particles and generate forces on these via the potential energy function. To do this, the volume is first cut into a set of tetrahedra, each of which contains four particles, one at each vertex. Each particle is included in one or several of these. Then, distance displacement functions are defined between a particle and its nearest neighbors, using the rest configuration for labeling. Then, a volume displacement function is defined using the coordinates of the four particles in each tetrahedron. Finally, a surface area displacement function is defined by measuring the area of the free surfaces of the tetrahedron and subtracting the rest area. One can then choose independent parameters for the *compressibility*, the *stiffness*, and the *surface tension*. This approach is an improvement over standard spring-and-damper models, where extra springs are introduced to model some of the material properties.

The first problem with this model is that these three parameters are not completely independent so that mapping to known material properties is difficult as discussed in [Nealen et al. 2005]. Essentially, the true deformation energy, that is easily measured in the lab and tabulated in handbooks, might not have such a simple mapping to the simple geometric quantities of the system of particles. This is very clear in the case of a string where definition of bending energy is difficult and torsion impossible, when considering only point particles.

The second problem is that forces generated by this model do not converge in the limit of infinite stiffness, even though the trajectories of the particles are mathematically well behaved in this limit. Indeed, as demonstrated in [Rubin and Ungar 1957], the penalty forces typically oscillate wildly in the limit where the stiffness constants become large. This is not an intuitive result since the constraint forces are well behaved in the case where the kinematic constraints $\phi_i(x) = 0$ are rigorously enforced. Therefore, it is generally difficult to integrate systems with large penalty forces. When using explicit integration methods, the time step is limited to less than a fraction of the smallest natural period of oscillations. In the present case, if all particles have identical masses m, this period is: $T_{min} = 2\pi\sqrt{m/k_{max}}$. This stability requirement seriously limits the

maximum stiffness or the performance. Some special implicit numerical integration methods improve stability but great caution is needed. For instance, when using the linearly implicit integration strategy of [Baraff and Witkin 1998] however, the stiffness restrictions are not so severe but artificial damping is clearly noticeable, limiting the usefulness of the method.

Two remedies are provided in the next section. The first is a definition of the potential energy of deformation which is not based on the simple geometric quantities but which correctly maps the strain tensor of elasticity to particle positions. This is similar to what is done in finite element methods. The second is a stable regularization of constraints so as to remove the high oscillatory components of standard penalty force formulations.

2.2 Constraints and regularization

Kinematic constraints impose restrictions on the motion of the particles in the system. For instance, we can simulate a particle moving on the plane $z = 0$ by imposing that restriction directly on the coordinate, thus bypassing the computation contact forces and acceleration due to gravity. It is well known in physics that kinematic constraints of the form[1] $\phi_i(x) = 0$ are the physical limit of strong potential forces of the form $(k_i/2)\phi_i^2(x)$, for very large k_i and this has lead to two main strategies for solving constrained system.

The first is to formulate the equations of motion taking the restrictions $\phi_i(x) = 0$ into account as is described in [Erleben et al. 2005] for instance. This requires the computation of *constraint forces* which are the solution of a non-linear system of equations. The main problems here are that even when linearizing, this system of equations can be computationally expensive to solve, and that the trajectories x tend to drift away from $\phi_i(x) = 0$ because of discretization and approximation errors. The common strategy for linearizing the equations for the constraint forces [Baraff 1996] and stabilizing the constraints $\phi_i(x) = 0$ [Baumgarte 1972] is notoriously unstable and this has given the constraint method much bad press. Stable and efficient methods for solving constrained systems do exist though and we will provide one of these in the next section.

The second strategy is to include the potentials $(k_i/2)\phi_i^2(x)$ and the corresponding forces. This latter approach is known as a penalty force computation. At the mathematical level, there is a rigorous correspondence on the trajectories produced by these two methods in the limit of infinite k_i. Given the simplicity of this formulation, penalty forces are very attractive. But this is deceptive. Indeed, a little known fact is that penalty forces do not converge to the smooth constraint forces corresponding to $\phi_i(x) = 0$. As shown in [Rubin and Ungar 1957], the penalty forces oscillate with very high frequency in the limit of large k_i. In technical terms, the convergence of the penalty forces is only weak∗. This means in particular that the average value of the penalty forces, over a short interval of time, Δt, say, does converge to the smooth constraint force. Numerically, this means that high penalty forces quickly generate instabilities which can only be resolved using special integration techniques designed for highly oscillatory systems. Some implicit integrators work well on highly oscillatory systems but some don't and a case of the latter is the first order implicit Euler method.

What we seek is a combination of the two strategies so that we can recover the stability of constraint computation but allow the modeling flexibility of penalty forces. To do this, we first state the equations of motion of the constrained system. We collect all the con-

---

[1]We consider only time independent equality constraints here. Inequality constraints may be used for non-penetration constraints, e.g., for collisions and contact.

straint terms in a vector function $\phi(x)$ with size n_c, the sum of the constraint dimensions. Of course, if $\phi(x) = 0$ at all times, we have $\dot{\phi}(x) = 0$ and by chain rule, this is $\dot{\phi}(x) = J\dot{x}$ where J is known as the Jacobian matrix of ϕ. The components of J are $[J]_{ij} = \partial\phi_i/\partial x_j$. Recall that for any surface defined with a scalar function, such as $\phi_i(x) = 0$, the gradient of $\phi_i(x)$ is normal to the surface, and that moving along the gradient is the fastest way to move away from the surface $\phi_i(x) = 0$. We want the constraint forces to directly oppose any force trying to move the system away from the surfaces $\phi_i(x) = 0$. It is therefore constructed as a linear combination of the constraint normals, $F_c = J^T\lambda$, where λ is a vector with n_c components, one for each constraint. What are the values of the components of λ? Well, just what is necessary to remove any part of resulting forces which point in the direction normal to any of the surfaces $\phi_i(x)$! Of course, since the point x moves in a direction *tangential* to any of the constraint surfaces $\phi_i(x) = 0$, the constraint forces we just defined are workless. When this is taken into consideration, and after writing $v = \dot{x}$ the *differential algebraic equations* of motion for the constrained system are:

$$\dot{x} = v \tag{3}$$
$$M\dot{v} = F_{ext} + F_{int} + F_c \tag{4}$$
$$\phi(x) = 0 \tag{5}$$

This is a nonlinear system of equation and a common strategy is to replace the constraint with a linear combination: $\ddot{\phi}(x) + a\dot{\phi}(x) + b\phi(x) = 0$, which is mathematically equivalent and stable if the coefficients a and b are positive. The resulting equation is linear in $\dot{v}$ as is easily verified, and the full system of equations (3)–(5) is then discretized as any other second order ordinary differential equation. This strategy is now getting known as the *acceleration based* method in contrast to so-called *velocity based* methods which we describe shortly. Note that this nomenclature is only used in the graphics literature. The problem here is that numerical stability is strongly dependent on the choice of the a and b parameters. However, there is no systematic strategy for choosing a and b which work in all cases. Choosing a and b too small leads to constraint drift so that $|\phi|$ increases with time, but choosing a and b too large makes the system explode. Conversely, a solution of the stabilized equations of motion with non-zero coefficients a, b, even when stable, is not necessarily a solution of the original system of equations. This method is a very bad idea indeed. Curiously, this is the most popular scheme in the graphics literature where it goes back to the early 1990s [Witkin et al. 1990].

The alternative is to discretize the velocity and acceleration *before* attempting to linearize the constraint $\phi(x) = 0$. This is covered in Sec. 2.4.

As was mentioned previously, constraints can be realized as the limit of strong potentials. If we keep the strength finite though large, this is a form of regularization if we find a way to write the equations of motion in terms of the inverse of the large stiffness. When discretized judiciously, this scheme can produce stable and efficient time stepping scheme of systems with either constraints or very strong forces. In fact, almost the entire range from 0 to infinity is allowed, although some elasticity may remain in the infinite case as a numerical error. Relaxing the constraints by keeping a finite but large penalty parameter (or small regularization parameter) has the added benefit of removing numerical problems which occur when constraints are degenerate or over defined, and to help stabilize the linear equation solving process by making the matrices strongly positive definite. In other word, we are trading the infinite stiff limit for speed and numerical stability.

Starting from a constraint ϕ we construct the potential energy:

$$U(x) = \frac{1}{2}\phi^T(x)\alpha^{-1}\phi(x) \tag{6}$$

for symmetric, positive definite matrix α of dimension $d_c \times d_c$. The correspondence to the penalty terms defined previously is the case where α^{-1} is a diagonal matrix and where the entries on the main diagonal are the stiffness parameters k_i. The limit $\alpha \to 0$ corresponds to infinite stiffness and $\alpha \to \infty$ to zero stiffness. In the case of distance constraints the corresponding potential is a spring potential, with spring stiffness α^{-1}. Since $U(x)$ is a potential energy term, it produces forces in the standard way, namely: $F_c = -\partial U/\partial x^T = -J^T \alpha^{-1} \phi$, where J is the Jacobian matrix of the functions $\phi(x)$ as before. Next, in order to replace the large parameters α^{-1} for the small α in the equations of motion, we introduce an artificial variable $\lambda = -\alpha^{-1}\phi$ such that $F_c = J^T \lambda$. For the regularized system the equations of motion are modified to:

$$\dot{x} = v \qquad (7)$$

$$M\dot{v} = F_{ext} + F_{int} + F_c \qquad (8)$$

$$\alpha\lambda(x,t) = -\phi(x,t). \qquad (9)$$

Note that in the limit of infinite stiffness, $\alpha \to 0$, the Eqs. (7)–(9) are free from singularities and reproduces the system (3)–(5). The trick now is to discretize this avoiding the high frequency oscillations in the constraint forces. Note also that a first order dissipative term of the form $-\beta\dot{\phi}$, with $\beta > 0$, can be also added to the right hand side of (9) without affecting the limit $\alpha \to 0$ as long as $\beta \to 0$ simultaneously.

2.3 Elastic deformation energies

Now that we have seen how to transform quadratic potential energy terms into constrained systems and vice versa, we construct a potential energy term for elastically deformable materials and define the functions $\phi(x)$ used in (6). Instead of using the intuitive geometric displacement functions $\phi(x)$ however, we turn to elasticity theory in order to approximate the strain tensor–a measure of deformation– in terms of the coordinates of the constituent particles. The benefit here is that all parameters entering the simulation are directly related to the known, tabulated material properties. Elasticity theory is rigorously covered in Ref. [Fung and Tong 2001] and more accessible for the purpose of physics based animation in Ref. [Erleben et al. 2005].

We will consider only linear (Hookean) and isotropic elastic materials here. This means specifically that the relation between the deformation and the restoring force is linear and therefore, the potential energy is quadratic in the deformations. To discretize the deformation and thus express it in terms of the particle coordinates, we use a spatial discretization found in *finite element analysis*, restricting our choice to linear *shape functions* and tetrahedral meshes. The mass is lumped at the nodes which correspond to point particles.

To parametrize the deformation of a solid, we first consider that it occupies some domain B in 3D space, in its rest configuration. Considering infinitesimal displacements first, each point $\mathbf{r} = (r_1, r_2, r_3)^T$ is moved by a small amount, $\mathbf{u}(\mathbf{r})$, so that its new location is: $\mathbf{r}' = \mathbf{r} + \mathbf{u}(\mathbf{r})$. This defines the vector field $\mathbf{u}(\mathbf{r})$ over the domain B. The field is needed for constructing a measure of deformation which is estimated by measuring the change in distance between two nearby points.

We assume that B is divided into a set of tetrahedra in what follows and concentrate the analysis on a single tetrahedron composed of four nodal particles with current positions $\mathbf{x}^{(a)}(t), a = 1, 2, 3, 4$.

The domain B is now the original, undisplaced, undistorted volume of our tetrahedron. To construct a mapping which relates the vector field $\mathbf{u}(\mathbf{r})$, to the *current* positions of the nodes, we need to compute the current displacement of each node as well as an interpolating shape function. The role of the shape function is to distribute the displacements at the nodes to displacements at the interior points in B so that if the node a_1 has a large displacement but node a_2 has none, the displacement field increases smoothly between these two nodes. We define $\mathbf{u}^{(a)}$ to be the full displacement vector of nodal particle (a) from an arbitrary initial position $\mathbf{x}_0^{(a)} = \mathbf{x}^{(a)}(0)$ where the tetrahedron is at rest, say, so we have: $\mathbf{u}^{(a)} = \mathbf{x}^{(a)}(t) - \mathbf{x}_0^{(a)}$. The simplest case is to use a linear interpolation so that if the vectors $\mathbf{u}^{(a)}$ are the nodal displacements at each node a, then, the displacement field reads:

$$\mathbf{u}(\mathbf{r}) = \sum_{(a)} N^{(a)}(\mathbf{r},x)\mathbf{u}^{(a)}, \qquad (10)$$

where summation is over the four particles in each tetrahedron, and x is the current configuration vector. Details of the full derivation of the shape function are found in [Erleben et al. 2005] for instance. The linear shape function is a scalar function of the form

$$N^{(a)}(\mathbf{r},x) = V^{-1}(a^{(a)} + b^{(a)}r_1 + c^{(a)}r_2 + d^{(a)}r_3), \qquad (11)$$

where $V(x)$ is the volume of the tetrahedra and the coefficients satisfy:

$$\begin{bmatrix} a^{(1)} & a^{(2)} & a^{(3)} & a^{(4)} \\ b^{(1)} & b^{(2)} & b^{(3)} & b^{(4)} \\ c^{(1)} & c^{(2)} & c^{(3)} & c^{(4)} \\ d^{(1)} & d^{(2)} & d^{(3)} & d^{(4)} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ x_1^{(1)} & x_1^{(2)} & x_1^{(3)} & x_1^{(4)} \\ x_2^{(1)} & x_2^{(2)} & x_2^{(3)} & x_2^{(4)} \\ x_3^{(1)} & x_3^{(2)} & x_3^{(3)} & x_3^{(4)} \end{bmatrix}^{-1} \qquad (12)$$

A few important observations must be made here. First, the vector $\mathbf{r}$ used in $\mathbf{u}(\mathbf{r})$ is in the interior of the *current* tetrahedron formed by the current coordinates of the nodes $\mathbf{x}^{(a)}$. Second, the displacement field $\mathbf{u}(\mathbf{r})$ is not small. Indeed, a uniform translation of all the nodes by a vector $\mathbf{y}$ produces $\mathbf{u}(\mathbf{r}) = \mathbf{y}$, which is arbitrarily large. A uniform rotation of all the particles also causes large changes in $\mathbf{u}(\mathbf{r})$. But the displacement field itself is not the measure of deformation. Instead, the Green strain tensor, which measures local variations in distances between close points $\mathbf{r}$ and $\mathbf{r} + \delta\mathbf{r}$, is what is needed. This is defined in terms of the *derivatives* of the displacement field as:

$$\varepsilon_{ij} \equiv \frac{1}{2}\left[\frac{\partial u_i}{\partial r_j} + \frac{\partial u_j}{\partial r_i} + \sum_{k=1,2,3}\frac{\partial u_k}{\partial r_i}\frac{\partial u_k}{\partial r_j}\right]. \qquad (13)$$

This tensor is symmetric and is therefore parametrized with a six dimensional vector: $\varepsilon = (\varepsilon_{11}, \varepsilon_{22}, \varepsilon_{33}, \varepsilon_{12}, \varepsilon_{13}, \varepsilon_{23})^T$. The quadratic term is often ignored but is necessary here if we want zero strain under rigid displacements.

The Green strain tensor is a good measure of small deformation but we use it for arbitrarily large ones. This can pose a problem if the four nodes collapse onto a plane, in which case, Eqn. (12) cannot be solved. Worse still, after going through a planar collapse, a tetrahedron can become *inverted* and eventually go to rest in an inside-out configuration. This can be remedied by adding an extra constraint to the system stating that the determinant of the matrix on the left side of Eq. (12) should be near unity but we do not pursue this further here.

Computing the derivatives of the displacement field is straightforward but tedious. The resulting expressions are found in [Erleben et al. 2005] for instance. The Ref. [Bro-Nielsen and Cotin 1996] is also useful for making implementations.

We now construct a potential energy in terms of the strain variables which are the natural measures of deformation. For an elastic material, the deformation energy per unit volume is given by $W(x) = \frac{1}{2}\varepsilon^T D\varepsilon$, where:

$$D = \begin{bmatrix} \lambda+2\mu & \lambda & \lambda & 0 & 0 & 0 \\ \lambda & \lambda+2\mu & \lambda & 0 & 0 & 0 \\ \lambda & \lambda & \lambda+2\mu & 0 & 0 & 0 \\ 0 & 0 & 0 & \mu & 0 & 0 \\ 0 & 0 & 0 & 0 & \mu & 0 \\ 0 & 0 & 0 & 0 & 0 & \mu \end{bmatrix}, \quad (14)$$

and λ and μ are the Lame constants which are directly related to other common material parameters such as Young's modulus, Y, bulk modulus B and Poisson ratio v, via simple algebraic relations, namely: $Y = \mu(3\lambda+\mu)/(\lambda+\mu)$, $B = \lambda + (2/3)\mu$, and $v = (1/2)\lambda/(\lambda+\mu)$. The matrix D is symmetric and positive definite as is well known. For large values of λ the material approaches incompressibility and with increasing values of μ the resistance to shearing deformations increases. Typical solids have Young's modulus ranging from 10 *MPa* for rubber to 2 *GPa* for Nylon, and up to 1000 *GPa* for diamond. The Poisson ratio ranges between 0 (compressible) and $1/2$ (incompressible). This means that λ and μ ranges between the order of the Young's modulus and infinity, $\lambda \to \infty$ latter being the limit of incompressibility.

We are now ready to formulate the kinematic constraint producing rigid body motion for a tetrahedron and the potential energy for elastic deformations. As the deformation energy $W(x)$ is an energy density, the potential energy for elastic deformation of a tetrahedron is the volume integral of this expression:

$$U = VW = \frac{1}{2}(V^{\frac{1}{2}}\varepsilon)^T D(V^{\frac{1}{2}}\varepsilon). \quad (15)$$

We identify the constraint for rigid body motion of a tetrahedron as $0 = \phi = V^{\frac{1}{2}}\varepsilon$ and the constant matrix α in the regularization description is identified as: $\alpha = D^{-1}$. We will employ this energy function for particle systems representing elastic deformable objects.

As noted previously, matrix D is constant, symmetric, and positive definite. It can therefore be factorized to the form $Q^T\alpha^{-1}Q$ where Q are constant orthogonal matrices and α^{-1} is diagonal with non-negative entries. The vector of constraints ϕ defined previously can therefore be associated with $\phi = Q\varepsilon$. Note here that the matrix Q essentially rotates the strain components, mixing them in a way that is dependent on the ratio between the Lamé constants. This mixing explains, in part, why simple geometric penalty function cannot be mapped easily to the physical parameters, even though the penalty terms are constructed from an identical tetrahedral mesh.

2.4 Time stepping

Details of numerical integration methods plays a critical role in simulation stability, accuracy–as well as the loosely defined notions of realism or plausibility–and even more so when using low order algorithms. The special objective we have in mind here is a stable integration of the highly oscillatory forces arising from the regularized constraint forces, combined with speed and a moderate level of accuracy. This contrasts with the numerical analysis literature where only *overall efficiency* is considered, namely, the accuracy achieved for a given unit of computational effort. In addition, accuracy is generally measured in terms of local or global error bounds on the solution itself. In simulating physical systems however, accuracy can also be measured in terms of preservation of known invariants of the trajectories such as momentum, energy, or

symplecticity of the flow, see Ref. [Kharevych et al. 2006]. Since the equations of motion are differential formulations of the consequences of these global invariants on the trajectories, numerical preservation of these invariants has a strong impacts on realism. For low order methods, the distinction is particularly obvious as we illustrate below. In fact, some low order methods, such as the popular symplectic Euler [Kharevych et al. 2006]–also known as Störmer-Verlet, Leapfrog, or several other names as well–*globally* preserve some of these invariants within certain bounds during the entire simulation, provided some stability restrictions on the size of the time step are met. By contrast, a fourth order Runge Kutta method, which is locally more accurate, does not preserve any of the physical invariants globally and can accumulate significant errors over time. For instance, with the wrong choice of time step, the energy of the system might decay to zero unintentionally–and inexplicably for the user.

We will discuss benefits and shortcomings of two widely used methods in the context of simulation of elastic materials first before introducing the regularized stepper which meets our objectives. Further examples and discussions are provided in section 3.

Consider the family of first order time steppers for the equations of motion (1)–(2):

$$x_{n+1} = x_n + \Delta t v_{n+n_x} \quad (16)$$

$$v_{n+1} = v_n + \Delta t M^{-1}F_{n+n_v}, \quad (17)$$

where the index n denotes the time point $t = n\Delta t$, and the indexes $n_x = 0,1$ $n_v = 0,1$ denote the discrete time at which the various quantities are evaluated. The explicit Euler method corresponds to the case $n_x = 0$, $n_v = 0$. It fails unconditionally on the undamped harmonic oscillator, producing trajectories with $|x_n| = O(e^{n\Delta t^2\omega^2})$, where $\omega^2 = k/m$. This is a bad sign. In general, artificial damping terms must be added to avoid exponential growth but careful analysis reveals that when the damping is increased too much, the system is *again* unstable. In addition, the observed damping forces are generally much lower than the damping parameters would suggest due to the in combination with damping–in general much larger damping than ever occurs in reality. Tuning this for a complicated network of spring and dampers is a tedious and frustrating task hardly worth the effort.

The case with $n_x = 1$, $n_v = 0$ corresponds to the well known symplectic Euler method which has many remarkable properties. For one, it is computationally as cheap as the explicit Euler method. However, it is stable for integrating harmonic oscillator systems as long as the time step satisfies $\Delta t < 2/\omega = 2\sqrt{m/k}$, without any artificial damping. It does not exactly conserve the energy but it does exactly preserve another quadratic form which is close to the energy. This implies global stability at least for linear systems as the energy oscillates within bounds proportional to Δt^2. The reason for this conservation property is that it is the simplest example of a variational integrator [Kharevych et al. 2006], and like all such time stepping methods, it is symplectic, i.e., it preserves an integral invariant along the flow. But since this is not unconditionally stable, we expect to lose stability as soon as $\omega_m > 2/\Delta t$, where ω_m is the maximum frequency in the system. This can therefore not be applied to the high stiffness case.

To understand this limitation, consider an object of mass 1 *kg* and density roughly like water. Simulate the object with 1000 particles of equal masses connected with spring forces with stiffness constant k, and integrate this with fixed time step $\Delta t = 0.02$ *ms*. The stability requirement demands that $k \lesssim 2.5$ *N/m*. The corresponding Young's modulus for this is roughly $Y \approx 250$ *Pa* which is extremely soft, as typical solids have Young's modulus ranging from 10 *MPa* up to 1000 *GPa*. At $Y = 250$ *Pa*, we are way off the scale. The

maximum stiffness become even less for a finer division, i.e., into more particles of lesser mass and shorter spring lengths. The way out is to integrate with smaller time steps or to use artificially small mass density, or to modify other physical parameters to achieve the desired visual result. But this only works if one is not interested in any sort of validation.

The case $n_x = 1$, $n_v = 1$ corresponds to the first order implicit Euler method. This is well known to be the most stable method in the book if the nonlinear system is solved accurately. It is also unconditionally stable for positive time step for the simple harmonic oscillator. When the nonlinear system of equations is solved approximately after a linearization, stability is no longer unconditional but still quite good. To do this, approximate the forces F_{n+1} with a Taylor expansion around the current state x_n, v_n to get the stepping formula:

$$x_{n+1} = x_n + \Delta t v_{n+1} \tag{18}$$

$$\left[1 - \Delta t^2 M^{-1} \frac{\partial F_n}{\partial x}\right] v_{n+1} = v_n + \Delta t M^{-1} F_n, \tag{19}$$

where the force derivatives are evaluated at time step n. The expression in the bracket on the left hand side of Eq. (19) we denote by $\mathbb{S}'_n$. This is a matrix of dimension $3N \times 3N$, that is typically sparse for the systems we are considering. For clarity we have discarded possible dependence on velocity in the force but this extension is simple to perform and well covered in the graphics literature. In particular the famous paper [Baraff and Witkin 1998] applies this to cloth simulations defined as networks of point masses attached with spring-damper. As observed in [Baraff and Witkin 1998], this is not unconditionally stable even for simple springs but stable enough to take much larger steps than would be possible with the symplectic Euler method, for instance. Solving the linear system of equations is not terribly expensive either and again, a good solution for speeding this up is provided in [Baraff and Witkin 1998].

We could of course consider this strategy to discretize the penalty forces or the regularization terms of (6). The problem lies elsewhere though. The implicit Euler method is stable because it artificially dissipates the energy of the system, faster with larger time step. On a simple linear harmonic oscillator, this artificial dissipation is so large that it significantly alters the observed oscillation frequency. Even worse, if one simulates a simple pendulum using a point mass attached to a stiff spring, integrating with the implicit Euler method alters the pendulum oscillation frequency and the observed acceleration of gravity, which should have nothing to do with the action of the stiff force! This artificial damping is immediately noticeable in cloth simulation, especially when comparison is made with energy preserving methods. This fact has not been reported in the graphics literature as an anomaly but is considered a welcome imitation of natural occurrences of friction although there is little control over it. We demonstrate how severe this damping can be in section 3.

When considering regularized equations of motion (7)–(9), we have additional variables to consider. Starting with the symplectic Euler parameters $n_x = 1$, $n_v = 0$ in (16–17), we still need to provide a correct interpretation of the regularized connection (9). We mentioned previously that penalty forces converge weakly to the constraint forces in the limit where $\alpha \to 0$. This means in particular that the time averages: $\tilde{\lambda} = (-1/\Delta t) \int_t^{t+\Delta t} dt \alpha^{-1} [\phi + \beta \dot{\phi}]$ should be well behaved. Formally integrating (8) over the time interval from t to $t + \Delta t$, where $t = n\Delta t$, we approximate $\int_t^{t+\Delta t} dt F_c \approx J_n^T \tilde{\lambda}$ and $\alpha \tilde{\lambda} = \Delta t^{-1} \int_t^{t+\Delta t} dt [\phi + \beta \dot{\phi}] \approx \Delta t^{-1} \phi_n + (1/2 + \Delta t \beta) J_n v_{n+1}$. Collecting the terms, we have:

$$x_{n+1} = x_n + \Delta t v_{n+1}$$
$$\begin{bmatrix} M & -J_n^T \\ J_n & \gamma \Delta t^{-2} \alpha \end{bmatrix} \begin{bmatrix} v_{n+1} \\ \Delta t \tilde{\lambda} \end{bmatrix} = \begin{bmatrix} M v_n + \Delta t F_n \\ -\gamma \Delta t^{-1} \phi_n \end{bmatrix}, \tag{20}$$

where $\gamma = (1/2 + \Delta t \beta)^{-1}$. This linear equation may either be solved directly using a sparse matrix solver, or by first building the Schur complement, $\mathbb{S}_n \equiv J_n M^{-1} J_n^T + \Delta t^{-2} \alpha$, and solve for the Lagrange multiplier from

$$\mathbb{S}_n \tilde{\lambda} = -\Delta t^{-1} J_n v_n - J_n M^{-1} F_n - \gamma \Delta t^{-2} \phi_n \tag{21}$$

and then compute the velocity from the top row of equation Eq. (20) and finally the positions are updated. The stability of this method has not been analyzed theoretically, but as we will see in the numerical examples below, it appears to be strongly stable. Computationally, it is no more expensive than the linear implicit Euler stepper. In fact, since we are skipping all Jacobian derivative terms which are used in the linear implicit Euler formulation, it is somewhat faster, and far, far simpler to implement.

3 An elucidating example

Consider a cube composed of eight particles and five tetrahedra, producing a total of 18 edges as shown in Fig. 2 a). We use this example to illustrate both the strength of the new formulation and statements made about numerical integrators in Sec. 2.4. As a model of elastic deformable material we assign energy distance functions for $U(x) = \sum_l (1/2) k \phi_l^2$ with $\phi_l \equiv (||\mathbf{x}_{(i_l)} - \mathbf{x}_{(j_l)}|| - L_l)/L_l$, where l labels each of the 18 edges in the mesh with respective rest lengths L_l, and i_l, j_l are the labels of the two nodes and the end of edge k. The cube is dropped from rest and subjected to downward gravitational acceleration of magnitude $g = 10 \ m/s^2$. One of the particles is fixed to the world so the cube swings back and forth at the same time as it undergoes elastic oscillations under its own weight. The mass of the particles are $m = 1 \ kg$ and stiffness is set to $k = 10^5 \ Nm$, yielding natural internal oscillation periods of 0.014 s. This is a nearly rigid case and the elastic deformation are expected to be small–of the order $\phi \sim 10^{-3}$–and hardly noticeable visually. We integrate the system with the three time steppers described in section 2.4 with time step $\Delta t = 0.05 \ s$, and for a reference solution with $\Delta t = 0.005 \ s$. The result is displayed in Fig. 2 and 3. The standard symplectic Euler stepper explodes almost instantly for $\Delta t = 0.05 \ s$ so only the first state is shown in Fig. 2 a). Fig. 2 b) displays snapshots from a simulation with the linear implicit Euler stepper and our new regularized stepper in Fig. 2 c), both for time step $\Delta t = 0.05 \ s$. The final position of the cube after a given number of steps is visibly different for the different steppers. In particular, the swinging motion is slower when using the linearly implicit Euler stepper than for the regularized one.

This is illustrated more precisely in Fig. 3 where the heights of a given particle is plotted as functions of time for all three methods using $\Delta t = 0.005 \ s$ and for the two stable methods for $\Delta t = 0.05 \ s$. A striking feature is that the *rate of fall* is slowed down by the linearly implicit Euler method when using the larger time step, taking nearly 35% more time to reach the minimum position, and this phenomena gets proportionally worse when either the time step or the stiffness is increased. The artificial damping of the integrator not only models linear drags in the direction of the spring forces but affects the physics of the *entire* system, even free fall under gravity! Worse yet, even when the time step is reduced by a factor of 10, down to less than one third of the natural frequency of the internal springs, damping is still of the order of 10% per cycle.

When using the smaller time step, the symplectic Euler stepper reproduces the same solution of as the regularized model. However, the regularized stepper nearly produces same trajectory for *both* time steps!
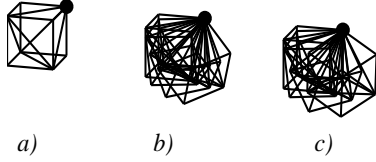
a)   *b)*   *c)*

Figure 2: An elastic cube anchored at one node, falling under gravity, is integrated with three different steppers: *a)* symplectic Euler stepper, that explodes almost instantly, *b)* linearly implicit Euler stepper and *c)* regularized stepper. The cubes are displayed at the time points $t = 0.0, 0.2, 0.4, 0.8\ s$ and integrated with large time step of $\Delta t = 0.05\ s$; nearly 3 times larger than the natural oscillation period of the structural springs. For the linear implicit Euler stepper numerical damping makes the cube fall slower than with the regularized stepper.
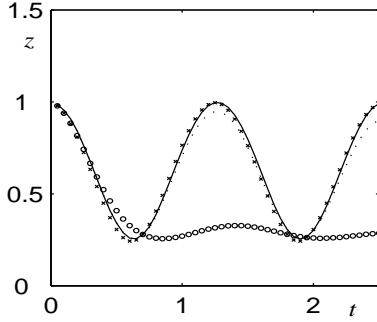
Figure 3: The height of one of the particles in the swinging cube as function of time. The severe damping of the linear implicit Euler stepper is clear. The motion is integrated with the linear implicit Euler stepper (circles for $\Delta t = 0.05\ s$, dotted line for $\Delta t = 0.005\ s$) and the regularized symplectic Euler stepper (crosses for $\Delta t = 0.05\ s$, solid line for $\Delta t = 0.005\ s$).

The observations are summarized as follows. For a given time step Δt, the symplectic Euler stepper is only stable as long as $k < 4m/\Delta t^2$. In other words, the time step must satisfy $\Delta t < T/\pi$ where T is the natural period of the oscillators, given by $T = 2\pi/\omega, \omega^2 = k/m$. For high stiffness, or short natural period, the solutions escapes to infinity. The linearly implicit Euler stepper is stable for much larger range of stiffness (which is difficult to determine exactly) but it adds artificial and spurious damping to the system. The result is that even the rigid body motion of very stiff materials are damped, producing noticeably inaccurate trajectories. The regularized symplectic Euler, on the other hand, produces an accurate trajectory even in the stiff limit and for large time steps. Some of this numerical behavior can be understood by comparing the matrices involved in the linear system of equations in Eq. (21) and (19), namely:

$$\mathbb{S}_n \equiv J_n M^{-1} J_n^T + \gamma \Delta t^{-2} \alpha \qquad (22)$$

$$\mathbb{S}'_n \equiv 1 - \Delta t^2 M^{-1} \frac{\partial F_n}{\partial x^T}$$

$$= 1 + \Delta t^2 M^{-1} \left(J_n^T \alpha^{-1} J_n + \frac{\partial J_n}{\partial x^T} \alpha^{-1} \phi_n \right) \qquad (23)$$

In the limit where $\alpha \to 0$, the first of these matrices has the nice limit: $\mathbb{S}_n \to J_n M^{-1} J_n^T$. This is the same matrix equation that must be solved to integrate the purely constrained system. However, the matrix defined in Eq. (23) goes to infinity. Looking carefully at matrix $\mathbb{S}'_n$, it can fail to be positive definite, numerically at the very least, even for moderately large values in α^{-1}, at which point the stepping is unstable and can diverge. As the identity term in $\mathbb{S}'_n$ becomes negligible in comparison with the α^{-1} terms, we get the wrong physics due to numerical errors in solving the badly scaled system of equations.

4   Simulating elastic deformable materials

We are proposing and investigating the combination of physically based material energies and constraint regularization technique for stable time stepping in simulations where the elasticity may range from very soft to very stiff. Here we present the details in constructing an actual simulation based on these ideas. We have identified the symplectic Euler stepper in combination with constraint regularization as a suitable numerical integrator. As mentioned earlier, length and volume constraints (e.g. spring-and-damper models) is a possibility, but is associated with an uncontrolled mixing of material parameters. This makes it impractical to adjust the parameters for the object to behave as a specific material. Instead we propose using constraints (or rather regularized with energy functions) based on elasticity theory, that have been described in section 2.3. We divide the object into a mesh of N_T tetrahedra and N nodes, or particles, as we did for the box in Fig. 2. Each tetrahedron has four node particles $(a) = (1), (2), (3), (4)$. For each tetrahedron we compute local quantities, e.g., strain and energy. These are added to global quantities. The *global* constraint vector ϕ and *global* Jacobian J then has dimensions $6N_T$ and $6N_T \times 3N$. The local quantities are constructed by first defining the *local* position vector $\tilde{x}^T = (\mathbf{x}^{(1)T}, \mathbf{x}^{(2)T}, \mathbf{x}^{(3)T}, \mathbf{x}^{(4)T})^T$ which is of dimension 12. We then define the local strain $\tilde{\varepsilon}$, that is of dimension 6 and in turn the local constraint deviation $\tilde{\phi}$ (also this of dimension 6) and local Jacobian $\tilde{J} = \partial \tilde{\varepsilon}/\partial \tilde{x}^T$, having dimension 6×12. The algorithm for the proposed method is

    initialize positions $x = x_0$ and velocities $v = v_0$
    construct tetrahedral mesh
    for each time step $n = 0, 1, 2, \dots$ **do**
       accumulate external forces
       loop {all tetrahedra}
          read off particles $(a) = (1), (2), (3), (4)$
          get local position vector $\tilde{x}$
          compute $\tilde{\phi} \equiv \tilde{\varepsilon}$
          compute $\tilde{J} = \partial \tilde{\varepsilon}/\partial \tilde{x}^T$
          add local $\tilde{\varepsilon}$ and $\tilde{J}$ to global matrix equation (20)
       end loop
       solve linear equation (20) $\to v_{n+1}$
       update $x_n \to x_{n+1}$
    end for

The global matrix equation here, can be either Eq. (20) or Eq. (21), depending on choice of linear equation solver. The most technical part is the computation of the local Jacobian and solving the linear system of equations. Some of the steps in computing the Jacobian are given in the Appendix.

5   Results

We now present results of validation and performance tests from an implementation of our new method.

28

5.1 Visual checks

Casual visual observations of the method in action reveal several nice properties, even with just a few tetrahedra. Still frames of simulation are shown in Fig. 4, where a beam made of 88 nodes and 105 tetrahedra is deformed by the action of a twisting force. The method is clearly capable of handling large deformations with a visually pleasing result, i.e., without kinks of collapsing regions. The same beam is illustrated in Fig. 5 in which the *nonlinear* terms in the Green strain, Eq. (13), were omitted. These terms are conventionally omitted when the displacements are only small. Such a linearization, whenever possible, largely improves the computational efficiency of the method. In that case the matrix in Eq. (20) is constant and its inverse may be precomputed to allow fast velocity updates. In the case of Fig. 5, the deformation is clearly too large for omitting the nonlinear terms. Otherwise, the result is an unrealistic volume deformation. We also show results for a sup-
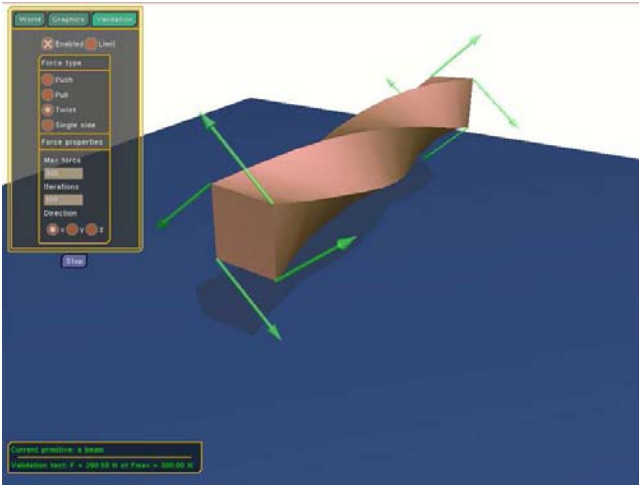

Figure 4: The result of a twisting force acting on the short ends of an elastic deformable beam.
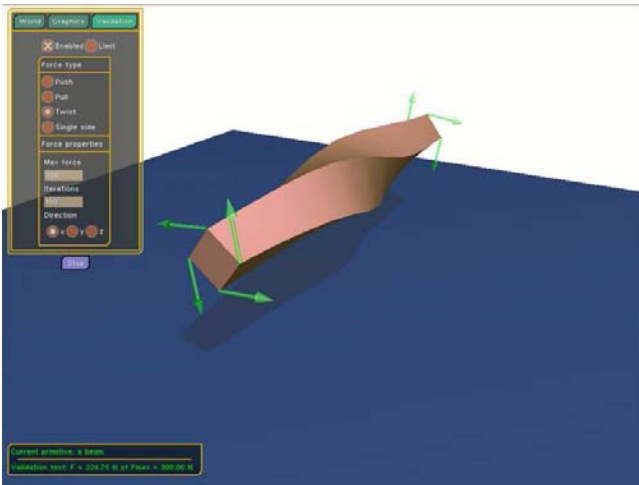

Figure 5: The same setup as in Fig. 4, but with the nonlinear contribution to the strain omitted. The result is an unrealistic volume deformation.

ported beam bending under gravity in Fig. 6 (with mass 100 *ton*)

and Fig. 7 (with mass 900 *ton*). In both cases we have Young's modulus $Y = 1.0$ *GPa* and the Poisson ratio is set to $\sigma = 0.25$. As expected, the material responds with stiffness increasing proportionally with the Young's modulus.
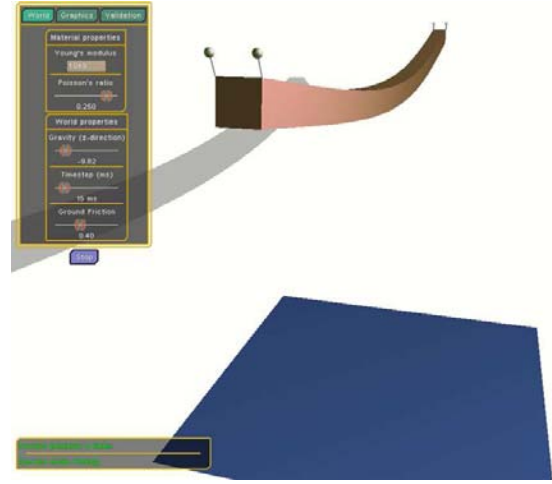

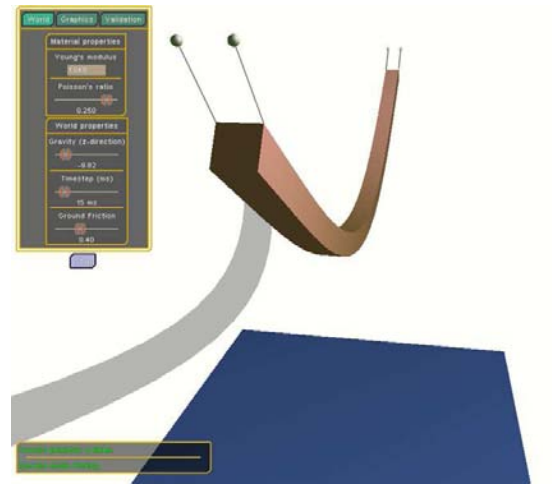Figure 6: A supported beam of mass 100 *ton*, $Y = 1.0$ *GPa* and $\sigma = 0.25$.

Figure 7: A supported beam of mass 900 *ton*, $Y = 1.0$ *GPa* and $\sigma = 0.25$.

5.2 Physical performance

Next we validate the physical behavior by conducting pull/push and twist tests and measuring the change in length and rotation for given applied forces. We validate against the theoretical relations between applied force and deformations from elasticity theory, i.e., Hooke's law for pull/push $\Delta L/L = F/YA$ and twist $\theta = 2\tau L(1+\sigma)/YK$, where L is the rest length, F is the applied force, A is the cross-section area, τ is the twisting moment, L is the length and K is a geometrical factor (torsion section constant). The result of these tests performed on a beam of 88 nodes and 105 tetrahedra are shown in Figs. 8 and 9. The results fit theory, but with a clear dependence on geometry/mesh. This is arguably, owed to the fact that we are

using 4-node tetrahedral mesh with *linear* shape function–simplest possible choice–which is known to be associated with particularly large mesh dependence. Using either finer discretizations or higher order shape functions should improve this. We also confirm that incompressibility is achieved for $\sigma = 0.5$.
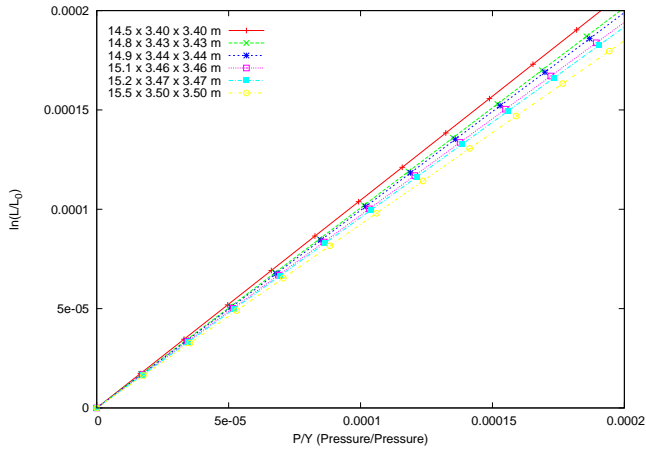

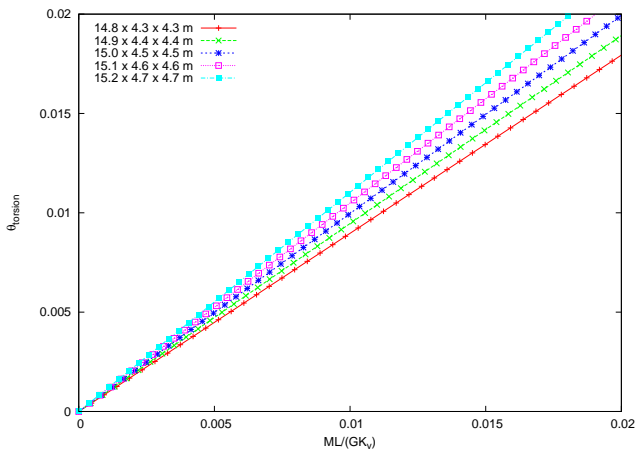Figure 8: The pull/push test for five slightly varying geometries.

Figure 9: The twist test for five slightly varying geometries.

5.3 Computational properties

We consider two aspects of computational performance: stability and efficiency. The regularized symplectic Euler integrator is self-stabilizing. This means that, much like the linear implicit Euler method, the simulation dissipates energy and approaches a state of equilibrium. It should be emphasized that the dissipation is restricted to the *internal motion* and does not affect rigid motion, which the linear implicit Euler method does, c.f. the swinging cube example in section 3. In real-time applications with time step $\Delta t = 0.015$ *s* the simulations are stable for stiffnesses even well above that of diamond $Y = 1000$ *GPa*. We achieve stable simulation of very stiff materials under large tension and large deformations. It should be pointed out, however, that the time scale of the dissipation of internal vibrations becomes very short for stiff materials. In Fig. 10 we show results from a soft beam attached in one

end and swinging freely under gravity. This figure confirms that the method is self-stabilizing. No energy dissipative terms have been added to the system, still, the system gradually relaxes to a state of equilibrium. The rate of numerical dissipation of the internal motion increases with decreasing size of the time step and with increasing stiffness. To resolve the issue of damped internal motion, the time stepper can be replaced with an energy preserving, though computationally more expensive, time stepper, e.g., a variational integrator as presented in ref. [Kharevych et al. 2006].
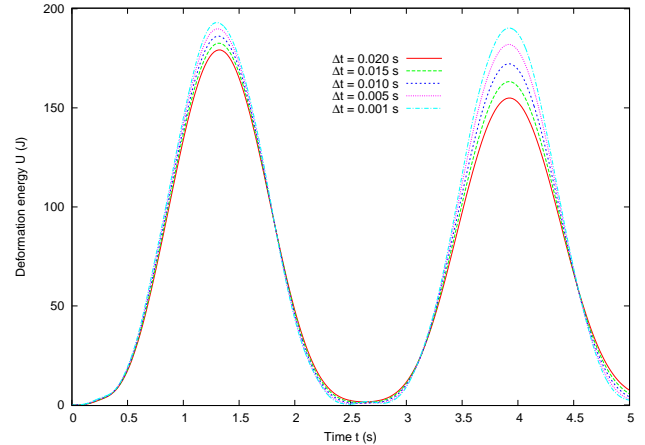

Figure 10: A beam is attached at one end and is swaying freely in a uniform gravitational field. This shows how the deformation energy vary over time at different sized time steps. Material parameters used are $Y = 8.0$ *kPa* and $\sigma = 0.25$ applied on the 10 kg beam.

Next we consider the efficiency of the method and how the computational time scales with system size. In this implementation we build and solve the Eq. (21) rather than Eq. (20) directly, making use of that the matrix is banded and achieve linear in time dependence on the number of particles in the system. The computational bottlenecks are recognized as the computation of the Jacobian and constraint, building and factorizing the matrix equation (21) and solving the equation. The contribution of these three processes in a simulation of a beam is displayed in Fig. 11. The simulation was run on an Intel Pentium 4 2.4 GHz CPU, with 1025 Mb DDR2 RAM internal memory. Most time, in our implementation, is consumed on factorizing the Schur complement matrix rather than solving it. The method we present clearly scales linearly with system size and real-time simulation (with time step of 15 ms) can be achieved for systems with size up to nearly 200 tetrahedra. There are several ways to improve the efficiency further, besides more efficient building and factorization of Eq. (21). In the computation of the new velocity, from Eq. (20), it is not necessary to build the Schur complement and first solve the Lagrange multiplier. Although Eq. (20), is larger in dimension than Eq. (21), it is a saddle-point matrix with a well ordered and sparse structure. There are efficient techniques, with linear complexity, for factorizing the matrix and solving this equation. The efficiency may be further increased using vector/parallel hardware, e.g., performing computations on the graphics processing unit. We estimate that with these optimizations real-time simulation of systems of the size of 1000 tetrahedra should be possible with currently available hardware.
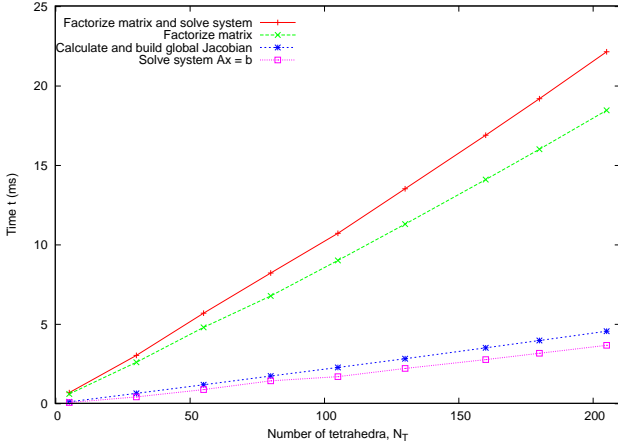
30

Figure 11: The computational time for the bottlenecks for a time step and the dependence on the number of tetrahedrons. The method we present has linear in time complexity and real time simulation is achieved with system size of about 200 tetrahedra.

6 Summary and Conclusions

We have constructed and investigated a method for stable simulations of elastically deformable objects at interactive rate with elasticity ranging from very elastic to highly stiff. With the novel combination of constraint regularization based on realistic material energies in combination with a symplectic Euler stepper we achieve stable time stepping for any value of stiffness of practical use, including the incompressible limit. This approach is compared to using linear implicit Euler, which damps all motion strongly in this limit. Any of the conventional integration methods for soft materials, including the standard symplectic Euler, has an upper limit on material stiffness for each size of time step - beyond this value the simulation becomes unstable.

Energy functions based on elasticity theory gives a direct relation between simulation parameters and real world measured material parameters. Simulated objects respond as expected to external forces and changes in the material parameters, e.g., Hook's law is obeyed and the material turns incompressible for the Poisson ratio $\sigma = 0.5$.

The method is self stabilizing. This guaranties stable simulation even in the regime of very stiff materials undergoing large deformations. On the downside, there is numerical, and thus artificial, damping in the system. Internal vibrations damp out very quickly for large material stiffness but rigid motion is unaffected. The numerical damping in our new method is thus less severe than for the linear implicit Euler method. The issue with numerical damping can be resolved by using time stepper that preserves this symmetry in the equations to a higher degree. These time steppers, known as variational integrators [Kharevych et al. 2006], are however more computationally expensive, they involve solving a non-linear system of equations rather than a linear system.

The method we present here scales linearly with system size. In the current implementation we achieve real-time performance of systems of the size of 200 tetrahedra. Performance can be improved by using other solver strategies, e.g., combining an iterative method and a good preconditioner, and utilizing vector/parallel hardware. We estimate that real-time simulation of systems of the size of 1000 tetrahedra should be possible with currently available hardware.

In comparison with other methods, it should be emphasized that the efficiency of the method we propose is *independent on the material stiffness* and not flawed by instabilities in the stiff regime.

In future work, we will improve the scaling and performance of the method, extend it to other type of constraints, e.g. contact constraints and improve the time stepping to reduce numerical dissipation without compromising stability and speed. Also, we will pursue issues of plasticity, mesh elements and adaptive level of detail techniques.

7 Acknowledgments

8 Appendix

Here we give some of the details in computing the Jacobian based on the energy function given in Eq. (15) from elasticity theory. Computing the Jacobian is part of building the equation 21. First, we order the strain vector ε in its *normal* components $\varepsilon_{normal} \equiv (\varepsilon_{11}, \varepsilon_{22}, \varepsilon_{33})^T$ and *shear* components $\varepsilon_{shear} \equiv (\varepsilon_{12}, \varepsilon_{13}, \varepsilon_{23})^T$ such that $\varepsilon = (\varepsilon_{normal}^T, \varepsilon_{shear}^T)^T$. We compute local quantities, e.g., the strain and contribution to the Jacobian for each tetrahedron. The local Jacobian is identified from the relation $\dot{\tilde{\phi}}_i = \tilde{J}_{ij}\dot{\tilde{x}}_j$, where $j = 1, 2, \ldots, 12$. Using $\phi = \varepsilon$ we identify

$$\tilde{J}_{ij}^{normal} = \Lambda_{iji} + \left(\frac{\partial u_k}{\partial r_i}\right)\Lambda_{kji} + \frac{\tilde{\varepsilon}_i^{normal}}{2\sqrt{V}}\frac{\partial V}{\partial \tilde{x}_j} \qquad (24)$$

$$\tilde{J}_{ij}^{shear} = \frac{1}{2}\Gamma_{imk}\Lambda_{mjk} + \frac{\tilde{\varepsilon}_i^{shear}}{2\sqrt{V}}\frac{\partial V}{\partial \tilde{x}_j}$$
$$+ \frac{1}{2}\chi_{imk}\left[\left(\frac{\partial u_n}{\partial r_k}\right)\Lambda_{njm} + \left(\frac{\partial u_n}{\partial r_m}\right)\Lambda_{njk}\right] \quad (25)$$

where

$$\Lambda_{ijk} \equiv \sqrt{V}\frac{\partial}{\partial r_k}\frac{\partial u_i}{\partial \tilde{x}_j}. \qquad (26)$$

In all of these expressions the ranges of the indexes are $i = 1, 2, \ldots, 6$, $j = 1, 2, \ldots, 12$ and $k, m, n = 1, 2, 3$. For notational convenience we have introduced the constant matrices χ and Γ that are both of size $3 \times 3 \times 3$ and with the nonzero elements

$$\chi_{112} = \chi_{213} = \chi_{323} = 1 \qquad (27)$$
$$\Gamma_{112} = \Gamma_{121} = \Gamma_{213} = \Gamma_{231} = \Gamma_{323} = \Gamma_{332} = 1 \qquad (28)$$

References

ALLEN, M. P. 2004. Introduction to molecular dynamics simulation. *N. Attig, K. Binder, H. Grubmller, and K. Kremer, editors, Computational Soft Matter: From Synthetic Polymers to Proteins, John von Neumann Instiue for Computing, Jlich, Germany,*.

BARAFF, D., AND WITKIN, A. 1998. Large steps in cloth simulation. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 43–54.

BARAFF, D. 1996. Linear-time dynamics using lagrange multipliers. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 137–146.

BAUMGARTE, J. 1972. Stabilization of constraints and integrals of motion in dynamical systems. *Computer Methods in Applied Mechanics and Engineering 1*, 1, 1–16.

BRO-NIELSEN, M., AND COTIN, S. 1996. Real-time volumetric deformable models for surgery simulation using finite elements and condensation. *Computer Graphics Forum 15*, 3, 57–66.

ERLEBEN, K., SPORRING, J., HENRIKSEN, K., AND DOHLMANN, H. 2005. *Physics-based Animation*. Charles River Media, Aug.

FUNG, Y. C., AND TONG, P. 2001. *Classical and Computational Solid Mechanics*. World Scientific, Singapore.

HAUTH, M., GROSS, J., AND STRASSER, W. 2003. Interactive physically based solid dynamics. *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*.

KHAREVYCH, L., WEIWEI, TONG, Y., KANSO, E., MARSDEN, J. E., SCHRDER, P., AND DESBRUN, M. 2006. Geometric, variational integrators for computer animation. *to appear in ACM/EG Symposium on Computer Animation 2006*.

LACOURSIÈRE, C. 2006. A regularized time stepper for multibody simulations. *Internal report, UMINF 06.04, issn 0348-0542*.

LANDAU, L., AND LIFSCHITZ, E. 1986. *Theory of Elasticity*. Pergamon Press, Oxford, 3rd ed.

MARSDEN, J. E., AND WEST, M. 2001. Discrete mechanics and variational integrators. *Acta Numerica 10*, 357–514.

MULLER, M., KEISER, R., NEALEN, A., PAULY, M., GROSS, M., AND ALEXA, M. 2004. Point based animation of elastic, plastic and melting objects. *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation 14*, 15.

MULLER, M., HEIDELRBERGER, B., TESCHNER, M., AND GROSS, M. 2005. Meshless deformations based on shape matching. *ACM Transactions on Computer Graphics (ACM SIGGRAPH 2005)*.

NEALEN, A., MULLER, M., KEISER, R., BOXERMAN, E., AND CARLSON, M. 2005. Physically based deformable models in computer graphics. *Eurographics State-of-the-Art Report*.

RUBIN, H., AND UNGAR, P. 1957. Motion under a strong constraining force. *Communications on Pure and Applied Mathematics X:65-67*.

TERZOPOULOS, D., PLATT, J., BARR, A., AND FLEISCHER, K. 1987. Elastically deformable models. *Communications on Pure and Applied Mathematics 21*, 205–214.

TESCHNER, M., HEIDELBERGER, B., MULLER, M., AND GROSS, M. 2004. A versatile and robust model for geometrically complex deformable solids. In *CGI '04: Proceedings of the Computer Graphics International (CGI'04)*, IEEE Computer Society, Washington, DC, USA, 312–319.

UMFPACK. `http://www.cise.ufl.edu/research/sparse/umfpack/`.

WITKIN, A., GLEICHER, M., AND WELCH, W. 1990. Interactive dynamics. *Computer Graphics 24*, 2, 11–21.

Computer visualization in accident prevention in industry

Sławomir Bogacki
Geometry and Engineering Graphics Centre
The Silesian University of Technology in Gliwice

Summary

The paper presents results from the collaboration between Polish coal mines and the Geometry and Graphics Center at the Silesian University of Technology in Gliwice, concerning the application of computer visualization in accidents prevention. The collaboration consisted of making movies which reconstruct accidents which have happened in coal mines. These movies are shown to employees during their health and safety training and on multimedia screens located in the plant. Further work carried out in our center will be concerned withe the development of an interactive simulator with which the employees would be able to learn safe job procedures .

1. Introduction

For the last 5 years, the number of accidents in Polish mines has stayed on the same level. On average, the number of deadly accidents amounts to 25 annually, which is unacceptable for both social and economic reasons. The main reasons of these accidents are still man faults. The mines' management is currently looking for new ways of appealing to employees' consciousness as far as work safety is concerned. It is mainly done through a change in a form and methodology of employee trainings [ILO 1993].
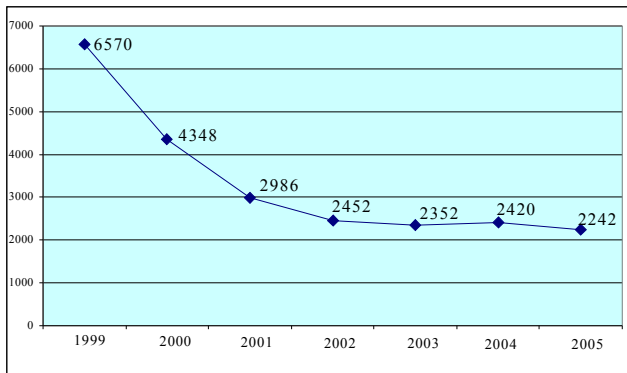

Fig. 1. The number of accidents in polish mines in last years.

For two years our center has been carried out work aimed at creating a series of animated movies which are use for training purposes and promoting safety among employees [Bogacki 2005].

Computer visualization is the best method for accident reconstruction due to the following reasons:

- High costs connected with employing actors and hiring TV equipment.

- Limited possibilities of using recording equipment (lighting, cameras) under ground, in coal mines.

- Reconstruction of the majority of the situations using traditional methods is not possible due to the high risk of the dangerous situations involved in an accident.

- It is impractical to make a video since work needs to be suspended at the location during the recording of the video.

The usage of computer visualization allows the reconstruction of even the most dangerous of situations. There is a possibility of modeling such events as tremors, inrush of water, methane or coal dust explosion.

The visualization will show the reaction of a crew. It is possible to reconstruct positions of their bodies and their actions. The scene can be watched from any direction so it is possible to simulate the field of view of each person involved in the accident. By using a computer generated visualization it is possible to determine what each person could see or even what he could not see. Observations done through the eyes of a casualty gives better understanding of the decision-action process which accompanied the analyzed events. Thanks to this feature of the computer generated visualization it is possible to use the visualization to verify the statements of witnesses to the accident .
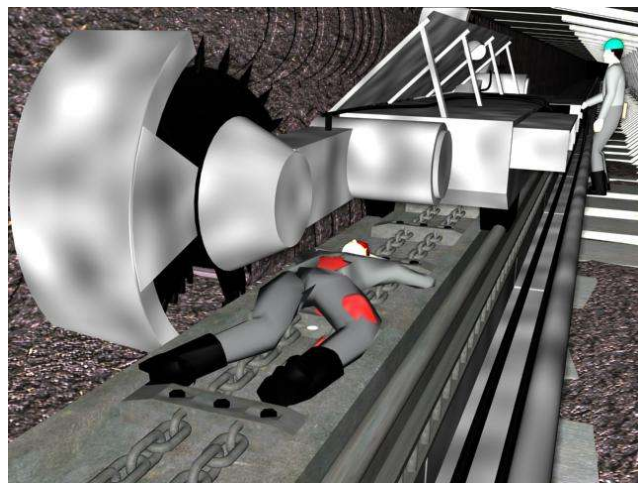

Fig. 2. Scene from movie, which reconstructed the accident in coal mine.

2. Assumptions

In order for a movie to fulfill its didactic purposes and for the employees to identify themselves with the presented situations the following issues must be included:

- A movie should realistically show the place where the accident happened, the machines, appliances and devices should look realistic, in a way known to the employees.

- The ambient conditions must be taken into account (lightning, dust).

- The movements of people and machines must correspond.

- Movie screenplay must be based on real events and the movie itself should focus on the error which was the main cause of the accident.

3. Production

The screenplay of the visualization needs to be based on the investigation that was carried out at the mine. Particular care is placed on witnesses' statements, what they witnessed and when, their location on the spot and their actions. The most convenient notation for the screenplay is storyboards since the plot of the visualization mainly consists of the movements of devices and people, and rarely features dialogues .

By means of visualization employees behavior is illustrated, their particular movements and activities. When each take is constructed, it must be cleary stated, minute by minute what movements the people and devices have made, including detailed information about their individual positions [Jacinto 2002].

The scene is reconstructed from of drawings and pictures of the location. All this data allows a detailed model to be constructed including all technical equipment involced in the situation. In order to simplify the creation of future visualizations, a set of 3d models have been created. These models represent machines and devices that are commonly used in polish mines. They are harvesters, transporting machines, as well as casing elements; however, for accidents reconstruction they do not have to be very detailed. In fact these machines are very complex and modeling of all their details would be a big drawback in scene operation and it would significantly increase the time of animation rendering. The immersion is increased when texturing and lighting of elements matches their real-world counterparts. This phase of the construction of visualizations has a big influence of the acceptance of the visulalization.
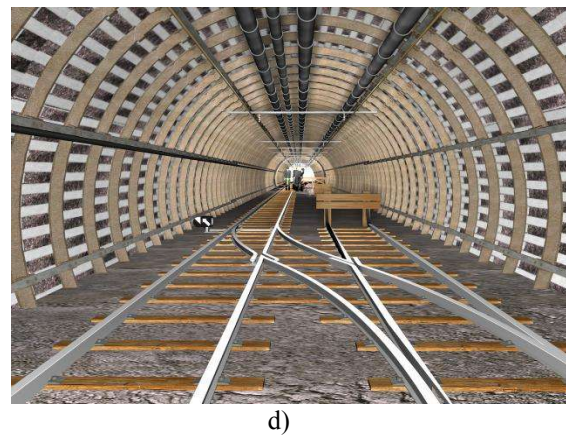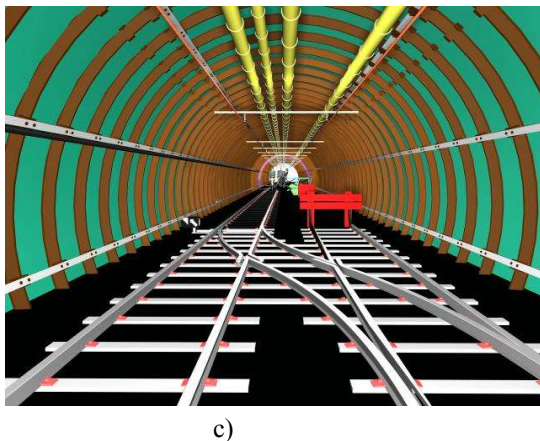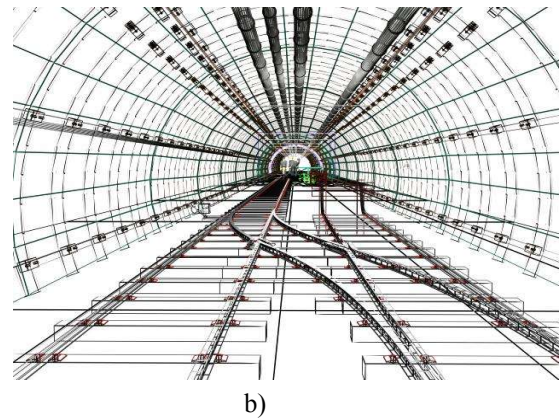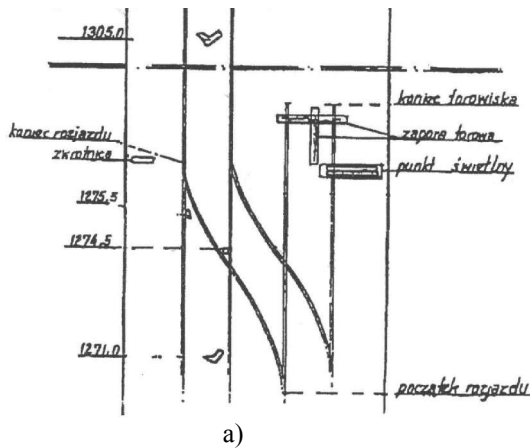

Fig. 3. Creation of spatial model of heading (railway crossover): a) sketch from accident's records, b) spatial mesh of heading, c) block model of heading, d) view of heading covered with textures.
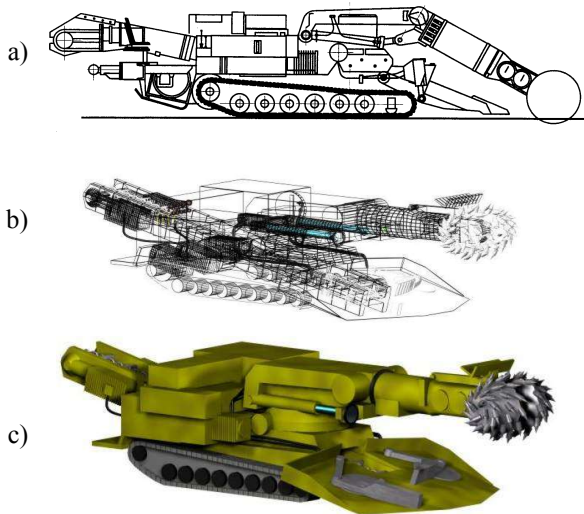
a)

b)

c)

Fig. 4 Creation model of harvester AM 50 a) side view, b) spatial mesh of harvester model, c) model of harvester covered with textures.

For the purpose of modeling, animation and rendering 3ds max program has been used. Animation of human figures has been done by means of 'biped' procedure from Character Studio module.
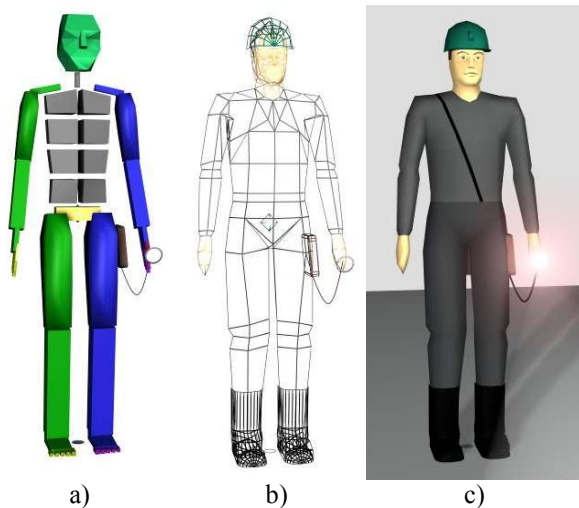

a)   b)   c)

Fig. 5. Stages of creation of miner profile a) skeleton of profile, used in movement generating, b) mesh of profile, c) profile covered with material.

Figures play the most important element in the movie but at the same time they cause the most troubles in animation. Other elements of equipment can be modeled with great realism whereas in case of human figures it is difficult to avoid superficiality, especially in the way they move. Nevertheless, for the purpose of events reconstruction some simplifications do not cause any obstacles. Further work on the visualizations will be required to increase the realism in the figures and movement of the employees .

The last phase of production is editing rendered material and adding sound corresponding to machines' noise. Narrator's commentaries are recorded separately, which describe particular situations. These commentaries are especially important when the movie is shown without a person leading the training.

4. Conclusions

From the experience of Health and Safety Departments it results that showing events reconstruction makes people realize how dangerous it is not to obey safety rules . It is especially appalling in case of dead bodies covered in blood.

Fig. 6. Scene from movie, which reconstructed the accident in coal mine.

Accident prevention should not be limited to the presentation of error consequences but it should also show how to deal with job procedures correctly, this includes emergency situations. Further works connected with accidents prevention is aimed at creating interactive simulator of production processes whose principle of operation will be similar to 3D games. This simulator will be used for teaching how to deal with job procedures safely in both typical and unplanned situations. Currently, work is being carried out on the development of a module for the creation of 3d models for visualization of industry accidents. The rendering engine from the open source Blender Modeling [Roosendaal and Selleri 2004] system is being adapted to fit the needs of the simulator .

References

BOGACKI S.: *Accidents reconstruction*. SEP Szczyrk 2005.

JACINTO C.: *Work Accidents Investigation Technique*. School of Engineering - MME, The University of Birmingham, May 2002.

INTERNATIONAL LABOUR OFFICE *Accident Prevention*. A Workers Education Manual. Geneva, 1993.

ROOSENDAAL T. SELLERI S.: *The Official Blender 2.3 Guide* Stichting Blender Foundation 2004

35

Incremental Spherical Interpolation with Quadratically Varying Angle

Anders Hast [*]
Creative Media Lab,
University of Gävle

Tony Barrera [†]
Barrera Kristiansen AB

Ewert Bengtsson [‡]
Centre for Image Analysis
Uppsala University

Abstract

Spherical linear interpolation has got a number of important applications in computer graphics. We show how spherical interpolation can be performed efficiently even for the case when the angle vary quadratically over the interval. The computation will be fast since the implementation does not need to evaluate any trigonometric functions in the inner loop. Furthermore, no renormalization is necessary and therefore it is a true spherical interpolation. This type of interpolation, with non equal angle steps, should be useful for animation with accelerating or decelerating movements, or perhaps even in other types of applications.

1 Introduction

Spherical linear interpolation (SLERP) [Glassner 1999] has got a number of important applications in computer graphics, for instance in animation [Shoemake 1985]. SLERP is different from linear interpolation (LERP) in the way that the angle between each vector or quaternion [Shankel 2000] will be constant, i.e. the movement will have constant speed. In the following text we will refer to quaternions even though everything presented can be used for vectors of any dimension. Nevertheless, the most probable application is for animation and here SLERP is used to interpolate quaternions.

LERP requires normalization and will also yield larger angles in the middle of the interpolation sequence. This will cause animated movements to accelerate in the middle, which is sometimes undesirable [Parent 2002]. However, if we can control the acceleration it should be useful in some cases since movements does not always have equal speed. There is already something called Spherical Quadratic Interpolation [Watt 1992], which involves quadratic interpolation between two linear interpolations and is therefore something completely different from what we propose herein. In our case we will have a quadratically varying angle, ie. the step size will will increase linearly.

The formula used for SLERP is

$$q(t) = q_1 \frac{\sin((1-t)\theta)}{\sin(\theta)} + q_2 \frac{\sin(t\theta)}{\sin(\theta)} \qquad (1)$$

where $t \in [0,1]$, and θ is the angle between q_1 and q_2 computed as

$$\theta = cos^{-1}(q_1 \cdot q_2) \qquad (2)$$

Note that q can be a quaternion or a vector of any dimension as explained above.

---

[*] e-mail: aht@hig.se
[†] e-mail: tony.barrera@spray.se
[‡] e-mail: ewert@cb.uu.se

It has been shown that SLERP can be efficiently performed without any computation of trigonometric functions in the inner loop [Barrera 2004]. In this paper we will show that it is possible to compute SLERP with a quadratically varying angle in a similar way. This type of interpolation should be useful for cases when the speed of the movement is close to quadratic, i.e accelerating or decelerating. The movement will be quadratic and the angle between two intermediate quaternions will increase linearly.

In [Barrera 2004] a number of alternative approaches are discussed that give efficient computation in the inner loop. The fastest one is the approach which uses Chebyshev's recurrence [Barrera 2004] but it requires equal angle interpolation. Another approach uses the De Moivre's formula [Hast 2003] and we shall concentrate on this approach in this paper since it can be changed to a quadratically varying angle interpolation quite easily.

1.1 Fast Incremental SLERP

The equation (1) can be rewritten as

$$q(n) = q_1 \cos(nK_\theta) + q_o \sin(nK_\theta) \qquad (3)$$

where q_o is the quaternion obtained by applying one step of Gram-Schmidt's orthogonalization algorithm [Nicholson 1995] and then it is normalized. The following code in matlab performs this operation and is assumed to come before the following code examples.

```
qo=q2-(dot(q2,q1))*q1;
qo=qo/norm(qo);
```

The quaternion q_0 is orthogonal to q_1 and lies in the same hyper plane spanned by q_1 and q_2. Furthermore, if there are k steps then the angle between each quaternion is $K_\theta = \frac{\cos^{-1}(q_1 \cdot q_2)}{k}$.

```
theta=acos(dot(q1,q2));
kt=theta/k;
q(1,:)=q1;
```

The code that follows on these three lines can be used for computing incremental SLERP. Usually SLERP is computed using trigonometric functions in the inner loop in the following way

```
b=kt;
for n=2:k+1
  q(n,:)=q1*cos(b)+qo*sin(b);
  b=b+kt;
end
```

However, by applying the De Moivre's formula [Nicholson 1995] it can also be computed efficiently using complex multiplication in this way

```
Z1=cos(kt)+i*sin(kt);
Z=Z1;
for n=2:k+1
  q(n,:)=q1*real(Z)+qo*imag(Z);
  Z=Z1*Z;
end
```

We can split the complex multiplication for each step into two different steps. But first we shall discuss quadratic interpolation in general in next section.

1.2 Quadratic Interpolation

We will start by showing how the quadratic interpolation for SLERP can be setup. Then in next section we will show how it can be computed efficiently. In [Hast 2003] it is shown how a quadratic interpolation can be setup so that it is computed by 2 additions only as it was proposed for shading by Duff [Duff 1979].

It is shown that a quadratic recurrence

$$t(n) = An^2 + Bn + C \tag{4}$$

where $n = [1..k]$ can be evaluated as

$$t_{i+1} = t_i + dt_i \tag{5}$$

$$dt_{i+1} = dt_i + d^2t \tag{6}$$

where $dt_0 = A + B$, $d^2t = 2A$ and $C = 0$ since we always start from the beginning of the interval between the vectors or quaternions.

2 Spherical Interpolation with Varying Angle

We can now put together an algorithm that uses the De Moivre's formula to interpolate spherically with a varying angle. Let us first split the angle

$$\alpha = \sigma\theta \tag{7}$$

$$\beta = \theta - \alpha \tag{8}$$

where $\sigma \in [-1, 1]$ is a scaling factor that will affect the change in speed.

Now we need to split up θ so that we will have a recurrence going from 0 to θ using α and β in k steps. We can do this by

$$A = \alpha/k^2 \tag{9}$$

$$B = \beta/k \tag{10}$$

To prove that this is the right way to do it, we put this into equation (4) for $n = k$ and we get

$$t(k) = \left(\frac{\alpha}{k^2}\right)k^2 + \left(\frac{\theta - \alpha}{k}\right)k = \theta \tag{11}$$

We know how to interpolate the angle quadratically and we can perform the spherical interpolation as exemplified in the following matlab code

```
theta=acos(dot(q1,q2));
A=sigma*theta; B=theta-A;
A=A/(k*k); B=B/k;
d2t=2*A;
dt=A+B;
t=0;

for n=1:k
  t=t+dt;
  dt=dt+d2t;
  qn(n,:)=q1*cos(t)+qo*sin(t);
end
```
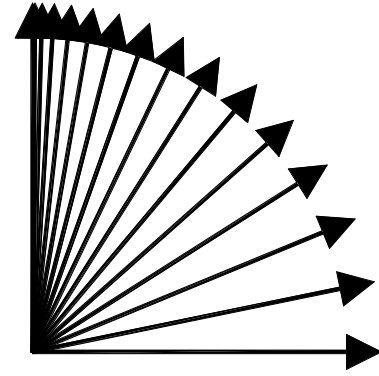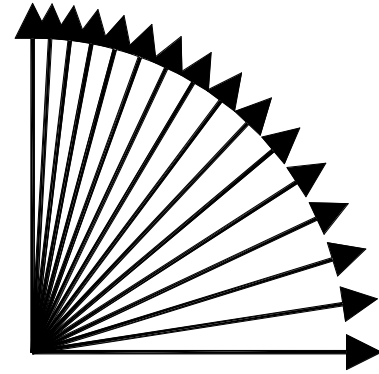

Figure 1: Interpolation with $\sigma = 1.0$.

Figure 2: Interpolation with $\sigma = 0.5$.

We can now set up the SLERP with Quadratically varying angle by using the approach recently explained and the code becomes

```
A=sigma*theta; B=theta-A;
A=A/(k*k); B=B/k;

Z1=cos(A+B)+i*sin(A+B);
Z2=cos(2*A)+i*sin(2*A);
Z=1;

for n=1:k
  Z=Z*Z1;
  Z1=Z1*Z2;
  q1*real(Z)+qo*imag(Z)
  qd(n,:)=q1*real(Z)+qo*imag(Z);
end
```

3 Results and Conclusions

We have shown how fast incremental spherical interpolation can be performed for a quadratically varying angle, which can be used for animation of accelerating and decelerating movements.

Figure 1 shows how the angle increases in the interpolation. Here $\sigma = 1.0$. Figure 2 shows that the rate of change can be modified by σ. Here $\sigma = 0.5$. If σ is very small then the approach becomes regular SLERP as shown in figure 3.

It is also possible to create a decelerating movement in a similar way by letting $\sigma < 0$. We have not investigated how the other
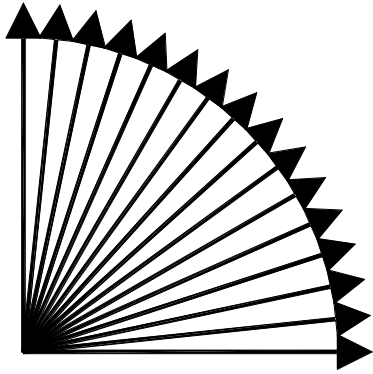
Figure 3: Interpolation with $\sigma = 0.000001$.

approaches for incremental SLERP can be modified for the varying angle approach but is something that should be done in the future. Hopefully this approach can be useful for animation and other applications. It is fast since no trigonometric functions need to be evaluated in the inner loop and the resulting quaternions or vectors will have unit length (if the starting and ending quaternions/vectors have unit length)

References

T. BARRERA, A. HAST, E. BENGTSSON 2004. *Faster shading by equal angle interpolation of vectors* IEEE Transactions on Visualization and Computer Graphics, pp. 217-223.

T. BARRERA, A. HAST, E. BENGTSSON 2005. *Incremental Spherical Linear Interpolation* SIGRAD 2004, pp. 7-10.

T. DUFF 1979. *Smoothly Shaded Renderings of Polyhedral Objects on Raster Displays* ACM, Computer Graphics, Vol. 13, 1979, pp. 270-275.

A. GLASSNER 1999. *Situation Normal* Andrew Glassner's Notebook- Recreational Computer Graphics, Morgan Kaufmann Publishers, pp. 87-97.

A. HAST, T. BARRERA, E. BENGTSSON 2003. *Shading by Spherical Linear Interpolation using De Moivre's Formula* WSCG'03, Short Paper, pp. 57-60.

A. HAST, T. BARRERA, E. BENGTSSON 2003. *Improved Shading Performance by avoiding Vector Normalization,* WSCG'01, Short Paper,2001, pp. 1-8.

W. K. NICHOLSON 1995. *Linear Algebra with Applications* PWS Publishing Company, pp. 275,276.

R. PARENT 2002. *Computer Animation - Algorithms and Techniques* Academic Press, pp. 97,98.

J. SHANKEL 2000. *Interpolating Quaternions* Game Programming Gems. Edited by M. DeLoura. Charles River Media, pp. 205-213

K. SHOEMAKE 1985. *Animating rotation with quaternion curves* ACM SIGGRAPH, pp. 245-254.

A. WATT, M. WATT 1992. *Advanced Animation and Rendering Techniques - Theory and Practice* Addison Wesley, pp. 363, 366.

A Driving Simulator Based on Video Game Technology

Mikael Lebram[*]
University of Skövde

Henrik Engström[†]
University of Skövde

Henrik Gustavsson[‡]
University of Skövde

Abstract

This paper presents the design and architecture of a mid-range driving simulator developed at the University of Skövde. The aim is to use the simulator as a platform for studies of serious games. The usage of video game technology and software has been a central design principle. The core of the simulator is a complete car surrounded by seven screens. Each screen is handled by a standard PC, typically used for computer games, and the projection on the screens is handled by budget LCD-projectors. The use of consumer electronics, standard game technology and limited motion feedback makes this simulator relatively inexpensive. In addition, the architecture is scalable and allows for using commercial video games in the simulator.

Observations from a set of experiments conducted in the simulator are presented in this paper. In these experiments driving school students were instructed to freely explore a driving game specifically designed for the simulator platform. The result shows that the level of realism is sufficient and that the entertainment value was considered to be high. This opens the possibilities to employ and use driving simulators for a wider set of applications. Our current research focuses on its use with serious games for traffic education.

CR Categories: I.3.2 [Computing Methodologies]: Computer Graphics—Graphics Systems; I.6.3 [Computing Methodologies]: Simulation and Modeling—Applications

Keywords: driving simulator, virtual reality, computer games, serious games

1 Introduction

The use of simulators for training is an old and well accepted method used in situations where training in real environments is difficult, dangerous and/or expensive. In particular, simulators for civil and military air pilot training are well established [Rolfe and Staples 1988]. Computer based flight simulators have been used since the 1960's. In the light of this, the usage of car driving simulators is less common. There exists advanced simulators for traffic safety research [Kuhl et al. 1995; ITS 2006; Östlund et al. 2006] and there are some examples of simulators for driving education [INRETS 2006], but for the vast majority of drivers the training is solely conducted in real traffic environments. The potential advantage of using driving simulators in, for example, traffic education

---
[*]e-mail: mikael.lebram@his.se
[†]e-mail: henrik.engstrom@his.se
[‡]e-mail: henrik.gustavsson@his.se

is that it enables generation of extreme situations or traffic environments not available in the student's surroundings.

Using simulators for drivers' education is different from pilot education in many ways. The volume of students involved in driving education is larger than that of pilot training. The cost associated with a driving training program is also much lower. This implies that simulators for drivers' education need to be less expensive in order to be widely adopted.

In the video game area, driving and racing has been a central theme for a long time. Since Night Driver (1976) [Atari 2006], the original first-person racing game, hundreds of racing titles have been released. In addition, car driving is a central activity in games of other genres as well. Over the last 30 years driving games has gone from relatively simple simulations in arcade-machines to highly realistic rally simulations that runs on an off-the-shelf personal computer (Figure 1).

Figure 1: 30 years of racing games. Night Driver [Atari 2006] (left) from 1976 and GTR II [SimBin 2006] (right) from 2006

Over the years the graphical quality of computer games has increased exponentially, the current level of detail of graphical models is sufficient for most training simulator purposes. In addition, video game developers have high skills in producing entertaining, immersive products that motivate their users to spend many hours a week [ESA 2005]. The games are sold as consumer products in large volumes which implies that the price is only a fraction of the price of a specialised simulator product.

The goal for the simulator presented in this paper is to utilize the developments in the video game area to create an advanced driving simulator using video game technology. This includes the use of standard of-the shelf soft- and hardware infrastructure as well as adaptation of commercial-of-the-shelf (COTS) games. The simulator is currently used to explore how serious games [LoPiccolo 2004; Zyda 2005; Blackman 2005] can be developed and used in traffic education.

This paper is organized as follows: In Section 2 we give a background presentation of driving simulator technology. In Section 3 we present the architecture and design of our driving simulator followed by Section 4 where we report experiences from using the simulator in a set of experiments. In Section 5 we draw conclusions and elaborate on the implications of our approach.

2 Background

Driving simulators are developed and used for research purposes mainly within the traffic safety context. The French national institute for transport and safety research presents a survey of simulators [INRETS 2006] including 50 simulators for research purposes, a handful for corporate purposes (e.g. car industry) and 20 driving training simulators. The research simulators are generally in the high-end with large budgets while the training simulators are typically low-range products targeted at driving schools.

A driving simulator is composed of a number of components, illustrated in Figure 2.
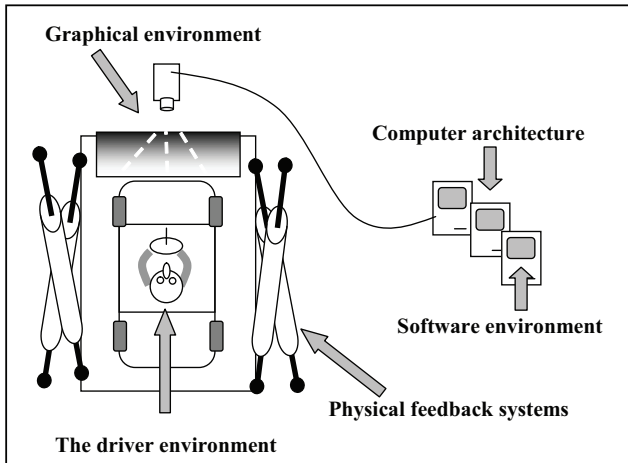

Figure 2: Components of a driving simulator

The driver environment may be more or less realistic ranging from an authentic complete car to a steering wheel (or even mouse and keyboard). The graphical environment renders the simulated world to the driver. It may differ in graphical quality, the size and range of the projection as well as the technology used to generate the view (CRT displays, 2D projection, 3D shutter-glasses etc.). The physical feedback system is responsible for generating other inputs to the driver, such as sound, motion and other haptic feedback. The computer architecture ties the various systems together and constitutes the platform for generating and monitoring the simulation by the software. The software available for a simulator is governed by the nature of the underlying systems. Specialised software may be needed if the underlying hardware is tailor made.

The most notable difference between high-end and low-end simulators is the physical feedback systems. The research simulators have large mechanical systems that generates g-forces in different directions. As an example, the *Leeds Advanced Driving Simulator* [ITS 2006] is a 4 million Euro project with a spherical projection dome. The driver environment and the dome are appended to a motion base that has 8 degrees of freedom. The simulator is hosted in a 14x12x7 meter hall. As a contrast *S-4150*, a basic training simulator from Simulator Systems International [SSI 2006], consists of a steering wheel and a CRT display in front of the driver. The simulator has clutch and brake and no physical feedback system. The cost of S-4150 is approximately 5 000 Euro.

In addition to the simulators mentioned above there exists a large number of "driving simulators" in the shape of video games. We refer to these as gaming simulators. A gaming simulator may be an arcade machine designed to give an entertaining experience and not necessarily a realistic one. There however exists gaming simulators

that are designed to produce a realistic driving experience. As an example, the GTR racing game has been developed by a racing team [SimBin 2006] with a goal to produce a highly realistic racing experience. For example, the simulation is so good that it is used by racing drivers to memorize courses [Björklund 2006]. The cost of a driving simulator composed of a PC, some CRT:s, a racing wheel and a game is typically less than 1 000 Euro.

3 The Driving Simulator

The goal of the driving simulator developed at the University of Skövde is to utilize the cost-effective and entertaining aspects from gaming simulators. The system is composed of 8 standard game PCs - 7 clients and a server. Each client is connected to a budget LCD-projector projecting on screens surrounding a real car. As the simulator is intended to be used for video games, the requirements on realism are somewhat different compared to high-end simulators used for traffic safety research. For games there is always an entertainment requirement that has to be considered. By using of-the-shelf hardware components it is possible to utilize game software and technology. We have successfully modified a number of COTS games to run on the simulator platform. In this section we present the architecture and design of the simulator.

3.1 The driver environment

The driver environment is a complete Volvo S80 with authentic controls and instrumentation. Figure 3 shows the driver environment.

Figure 3: The driver environment

The use of a real car provides a great deal of realism to the simulator. Users of the system have no problems in understanding the functionality of the interface to the simulator. In addition the feeling of being inside a car is a familiar situation which, for most people, is associated with a responsibility for the car and fellow road users. This brings a sense of seriousness to the driving.

3.2 Graphical environment

The graphics generated in the simulator is projected on seven flat screens as illustrated in Figure 4. These screens cover the whole field-of-view for the driver and the parts covered by the rear view mirrors. In the design phase an alternative solution, to back-project the graphics directly on the windshield, was rejected for a number

of reasons: Firstly this solution gives a very closed and flat projection where the external parts of the car are not visible. It is also not possible for the driver or passengers to move their heads to get a different perspective on the surroundings (e.g. if the windshield post is covering some part of the view). Another aspect to consider is the distance from the observer to the projected screen. The simulations are generated using window-projection [Cruz-Neira et al. 1992] that is computed from the perspective of the driver which means the passengers will experience a distortion. With the chosen solution the distance to the screens is greater and the distortion for passengers is acceptable. An additional advantage by using screens outside the car is that it enables the original rear mirrors to be used and it is possible for the driver to turn his head and look in the rear window. The latter is not possible when, for example, LCD-displays are used as rear mirrors.
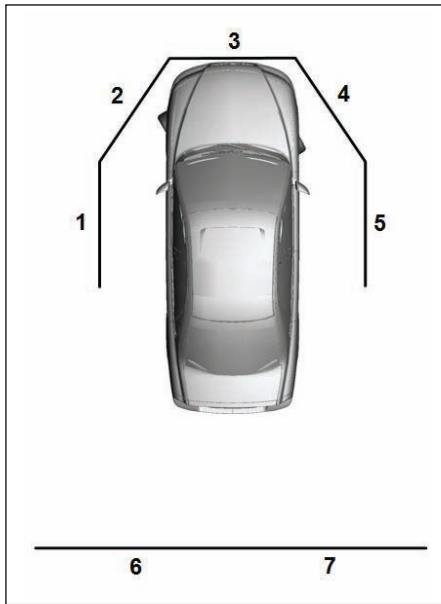

Figure 4: The simulator car surrounded by screens (1-7)

The projection on the screens is similar to that used in a Cave [Cruz-Neira et al. 1992]. The choice of rectilinear projection instead of cylindrical or spherical is mainly economical. Each screen is handled by a budget LCD-projector and as the screens are not projected seamlessly there are low requirements on the calibration. This also makes it possible to use a large number of screens and hence cover a larger fraction of the field-of-view, than is common in mid-range simulators. The forward visual field-of-view is 220 by 30 degrees, and 60 by 30 degrees in the rear direction. As a comparison, the high-end simulator used in [Peters and Östlund 2005] has a forward visual field-of-view of 120 by 30.

3.3 Physical feedback systems

The generation of physical feedback in a driving simulator may be extremely complex. The simulator at the university of Skövde adopts a fixed-based approach which means that no g-forces are generated. This is in total contrast with the mid-range simulator presented by Huang and Chen [Huang and Chihsiuh 2003] which emphasises on the motion system in favour of the graphical system and the driver environment.

The illusion of movement in our simulator is generated by the use of sound, vibrations and the car's fan. The sound is generated in the in-

ternal surround system of the car. In addition a "ButtKicker" [Guitammer 2005] is used to generate vibrations in the body of the car which are propagated to the whole car including the steering wheel. One important property relating to physical feedback in a driving simulator is the haptics of the steering wheel. In a car with servo-steering there is not as much movements as when there is a direct connection between the steering wheel and the tyres. The most important remaining physical property is that the wheels should strive to return to their original position. In the simulator this has been achieved by placing each front wheel on an axial ball bearing. Due to the steering axis inclination there will be a strive to return the wheels to a parallel position. In addition, the movements of the front wheels gives a notable movement of the car that can be considered to be a form of passive physical feedback.

The physical feedback component that possibly contributes most to the perception of speed in the simulator is the internal fan. It is controlled by the simulation and the force of the fan is linear to the speed of the car [Carraro et al. 1998]. When the driver is reaching a high speed the wind and the substantial noise from the fan contributes to create a high speed perception. It is well known that it is difficult to get a good perception of speed in computer generated simulations [Godley and Fildes ; Östlund et al. 2006]. The use of a fan is a simple but effective way to increase the perceived speed.

3.4 Computer Architecture

The computer architecture in the simulator consists of 8 standard gaming PCs equipped with a mid-range graphics card. One computer is acting as server while the other are clients each responsible for one screen. The clients and server are typically running identical simulations with the only difference that the server is sending synchronize messages to the clients. The clients differ only in the camera position used when rendering. The computers are connected in an Ethernet LAN. All hardware components are standard consumer products. The only tailored component of the simulator is the interface with the car [Mine 1995]. The movements of the steering wheel and other controls are monitored by microcontrollers that communicate with the server via a USB game control protocol. In this way the car can be seen as a highly specialised joystick. The advantage with this approach is that the simulator can be used with any computer game that supports joysticks.

3.5 Software Environment

As mentioned above, almost any computer game can be played on the simulator platform. In most situations it will however be limited to use only one screen. To utilize all 7 screens the software has to support multiple clients with adjustable camera positioning. The extensions required are hence very small and we have successfully managed to adjust several commercial games to be used in the simulator using multiple screens. In addition to using COTS games we have also developed an infrastructure based on an open source game engine. This allows for custom made simulation application and games.

4 Experiments

The simulator has been used in an experimental study with 24 driving school students as subjects. The gaming background of the students varied from inexperienced (13) to experienced players (5). The experimental setup was such that the subjects were offered as

much time they wanted (up to a maximum of 30 minutes) playing and exploring a game. This was followed by a number of evaluative tests where they were instructed to perform certain tasks, followed by a questionnaire. All simulations were monitored and logged.

The game used in these experiments is relatively simple. The player is driving on a five-lane motorway following an ambulance. The difficulty of the game increases by the intensity of the traffic and the behaviour of fellow road users. Although we had other main goals with these experiments, they have also provided some feedback on the performance of the simulation environment.

First of all, the subjects where extremely positive concerning the entertainment value of the simulator. In the questionnaire subjects were asked to specify how they agreed to the statement "it was fun to drive", on a 5-graded Likert scale where 1=fully disagree and 5=fully agree. The average for all subjects where 4.6 which is a very high result considering the relative simplicity of the game. This result may also be derived from the amount of time the subjects spent in the simulator. They were explicitly instructed to decide themselves when to stop driving. The result was that experienced gamers spent on average 29 minutes playing the game compared to 23 minutes for inexperience players. This is a statistically significant difference which is interesting as one may suspect that experienced gamers would not appreciate a game that is far from a state-of-the-art racing game. One possible interpretation is that the simulator platform itself contributed to the positive experience, in particular for gamers.

Concerning the realism of the game and simulator the average of the subjects was 3.6 for the statement "the driving was realistic" (using a 5-gradet Likert scale). This is clearly above average which indicates that the simulator is efficient. Some users commented on initial problems with the control of the car. These initial problems do not seem to have had any negative impact on the overall experience and performance of the drivers. This can be confirmed from analysis of how the drivers managed to position their car in the lane (lateral position). Lateral position is commonly used for validation of driving simulators [Green 2005; de Waard et al. 2005]. Figure 5, illustrates the relative lateral position of the car during all experiments.
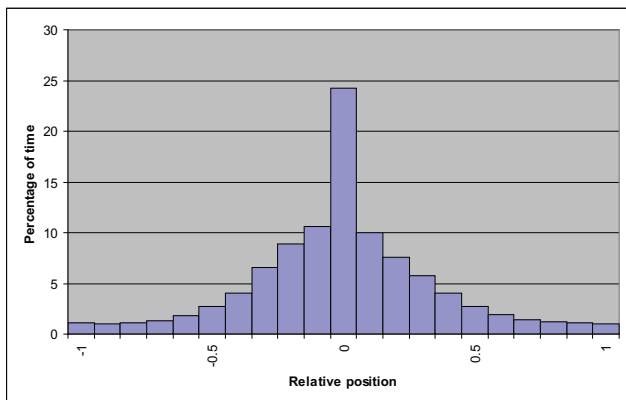

Figure 5: Histogram of the relative lateral position of the car

The total driving time for all subjects was almost 12 hours. The position of the car was sampled at 10Hz. The histogram in Figure 5 is based on all logs from all experiments (413 973 samples). The relative position of the car in the lane was divided into 21 discrete intervals. The histogram was created by summarizing the number of times the car was positioned in respective interval and then divide it with the total number of samples. Note that the recorded informa-

tion only considers the relative position within the lane (irrespective of what lane the car was in). The tails in the histogram are due to lane changes and the gameplay is such that frequent lane-changes are required to succeed. In fact, the drivers changed lane on average every 15 seconds. The most notable property of the histogram is the large bar in the middle. Despite the frequent lane changes the drivers spent almost 25% of the time exactly centred in the middle of the lane. The central bar is moreover more than double the size of the surrounding bars. We interpret this as the drivers have intuitively managed to position the car very close to the centre of the lane. This implies that the visual representation gives a realistic impression of the position of the car. The rectilinear projection hence seems to work very well.

The use of original rear view mirrors also seems to be efficient. The subjects' use of the mirrors was monitored during experiments and the result shows that they used both the internal as well as the external mirrors frequently. In fact, the use of mirrors was more frequent than the lane-changing. On average the subjects used the mirrors every 10 seconds compared to 15 seconds for lane changes.

Simulator sickness (also termed cybersickness) is a well known problem in simulators and is related to motion sickness [Harm 2002; AGARD 1988]. As much as 30% of the users of simulators may experience symptoms severe enough to discontinue use [Harm 2002]. Simulator sickness is believed to be caused by confusion between the perceived motion and the actual motion [Bertin et al. 2004]. The problem seems to be difficult to totally eliminate, even for high-end simulators [Peters and Östlund 2005].

Since the simulator presented in this paper is a fixed-based system, problems with simulator sickness was not unexpected. These problems were however minor in the experiments. Four subjects (17%) reported sickness as one of the reasons they decided to stop playing the game. The average playing time of these four subjects was 21 minutes compared to 25 minutes for those that did not report any sickness problems. The relatively small difference in time makes us believe these subjects did not experience severe problems with simulator sickness.

5 Discussion

In this paper we have presented a driving simulator based on video game technology. Our approach has been to use relatively inexpensive hardware components to create a graphical system that surrounds a real car whose instrumentation has been adopted to be used as a game control. A main difference to high-end simulators is the modest physical feedback system. The presented simulator uses a fan, vibrations and sound in addition to the graphical feedback. The driving simulator has successfully been used in an experimental study. Observations from this study indicate that the simulator is efficient in that it creates a realistic and entertaining experience to the users. The absence of physical feedback does not seem to incur serious problems with simulator sickness. In addition the rectilinear projection gives a realistic perception of the simulated environment. This has been shown by analysing the lateral positioning of the car.

When developing a simulator one goal is to create a realistic experience. Realism comes to a price and with a limited budget the benefit has to be balanced with its price. In our approach we have decided to sacrifice the physical movement realism in favour of the realism of having a real car as the driver environment. We believe that the use of a real car is one of the key benefits of the presented simulator. The smell and touch of a car gets the driver in the mind-set of driving. In addition, our simulator allows for passengers, which is

a typical driving property which is neglected in many other simulators. The driving task is, for example, much harder to handle when there are two fighting children in the backseat.

The simulator architecture presented in this paper is flexible and scalable. For example, the number of screens used can easily be extended by adding a projector and a PC for each screen. The server can broadcast messages to all involved clients which mean that the total load of the system is in practice independent of the number of screens. Each PC handles the rendering of one screen which differs only in their camera positioning. The flexibility of the architecture is illustrated by the fact that we have successfully modified several COTS games to be playable on the simulator platform. The ability to adopt and use commercial software is important as the cost of software development may be huge. In future studies we plan to use COTS racing games whose graphical quality require budgets way beyond that of the simulator hardware.

To summarize, the contribution of the presented work is that we have combined the quality and cost-effectiveness of the gaming technology with the extensiveness of the mid-range to high-end simulators. We estimate that the hardware cost of the presented simulator is less than 20 000 Euro excluding the cost of the car.

In our ongoing and future research we will use the simulator to explore the potential benefit of using computer games in traffic education. We will test whether a serious driving game designed with the specific purpose of enhancing certain traffic safety variables is effective. The general idea is to combine the strengths of traditional simulators with the fun of games.

Acknowledgements

References

AGARD, 1988. Motion cues in flight simulation and simulator induced sickness. Advisory Group for Aerospace Research & Development, Conference proceedings.

ATARI, 2006. Atari inc. Retrieved 2006-9-23, from http://www.atari.com.

BERTIN, R., GUILLOT, A., COLLET, C., VIENNE, F., ESPIÉ, S., AND GRAF, W., 2004. Objective measurement of simulator sickness and the role of visual-vestibular conflict situations: a study with vestibular-loss (a-reflexive) subjects. Neuroscience, San Diego, California.

BJÖRKLUND, P., 2006. Chief technical officer SimBin development team AB. Personal Communication.

BLACKMAN, S. 2005. Serious games...and less! *SIGGRAPH Computer Graphics 39*, 1, 12–16.

CARRARO, G. U., CORTES, M., EDMARK, J. T., AND ENSOR, J. R. 1998. The peloton bicycling simulator. In *VRML '98: Proceedings of the third symposium on Virtual reality modeling language*, ACM Press, New York, NY, USA, 63–70.

CRUZ-NEIRA, C., SANDIN, D. J., DEFANTI, T. A., KENYON, R. V., AND HART, J. C. 1992. The cave: audio visual experience automatic virtual environment. *Commun. ACM 35*, 6, 64–72.

DE WAARD, D., STEYVERS, F., AND BROOKHUIS, K. 2005. How much visual road information is needed to drive safely and comfortably? *Safety Science 42*, 7, 639–655.

ESA, 2005. Entertainment software association - game player data. Retrieved 2005-04-13, from http://www.theesa.com/facts/gamer_data.php.

GODLEY, S. T., AND FILDES, B. N. Driving simulator validation for speed research. *Accident analysis & prevention 34*.

GREEN, P., 2005. How driving simulator data quality can be improved. DSC 2005, Orlando.

GUITAMMER, 2005. Buttkicker. Retrieved 2006-10-02, from http://thebuttkicker.com/.

HARM, D., 2002. Motion sickness neurophysiology, physiological correlates, and treatment. in Handbook of Virtual Environments, Stanney (edt.).

HUANG, A., AND CHIHSIUH, C. 2003. A low-cost driving simulator for full vehicle dynamics simulation. *IEEE Transactions on Vehicular Technology 52*, 1, 162–172.

INRETS, 2006. Driving simulators. Retrieved 2006-10-02, from http://www.inrets.fr/ur/sara/Pg_simus_e.html.

ITS, 2006. Leeds advanced driving simulator. Retrieved 2006-10-02, from http://www.its.leeds.ac.uk/facilities/lads/.

KUHL, J., EVANS, D., PAPELIS, Y., ROMANO, R., AND WATSON, G. 1995. The iowa driving simulator: An immersive research environment. *Computer 28*, 7, 35–41.

LOPICCOLO, P. 2004. Serious games. *Computer Graphics World 27*, 2.

MINE, M. R. 1995. Virtual environment interaction techniques. Tech. rep., Chapel Hill, NC, USA.

ÖSTLUND, J., NILSSON, L., TÖRNROS, J., AND FORSMAN, A., 2006. Effects of cognitive and visual load in real and simulated driving. VTI rapport 533A-2006, Swedish Road and Transport Research Institute (VTI), Linköping, Sweden.

PETERS, B., AND ÖSTLUND, J., 2005. Joystick controlled driving for drivers with disabilities. VTI rapport 506A-2005, Swedish Road and Transport Research Institute (VTI), Linköping, Sweden.

ROLFE, J., AND STAPLES, K., Eds. 1988. *Flight Simulation*. Cambridge University Press.

SIMBIN, 2006. Simbin development team. Retrieved 2006-9-23, from http://www.simbin.se.

SSI, 2006. Simulator systems international. Retrieved 2006-10-02, from http://www.simulatorsystems.com.

ZYDA, M. 2005. From visual simulation to virtual reality to games. *Computer 38*, 9, 25–32.

The Verse Networked 3D Graphics Platform

Emil Brink[*]
Royal Institute of Technology

Eskil Steenberg[†]
Royal Institute of Technology

Gert Svensson[‡]
Royal Institute of Technology

Figure 1: Screenshot rendered by Quel Solaar, showing multiple materials, texture-mapping, shadows and per-pixel lighting.

Abstract

This paper introduces the Verse platform for networked multiuser 3D graphics applications. The client/server-oriented use of the network architecture is described, as is the highly heterogeneous client concept. The data storage model is introduced, as are several features of the object representation. A single-primitive approach to high-quality graphics based on a hybrid of Catmull-Clark and Loop subdivision surfaces is described, together with a high-level material description system.

Keywords: network, 3D, graphics, subdivision, open, platform, cooperation, distributed

1 Introduction

The Verse[1][Brink and Steenberg 2006] project was started at the Interactive Institute in June of 1999, in an attempt to make the threshold for working with networked 3D graphics applications lower. It was felt that existing tools and architectures either were not open enough, lacking in modern features, or both. The first version of the protocol was developed by Eskil Steenberg and Emil Brink.

[*]e-mail: ebr@kth.se

[†]e-mail:eskil@obsession.se

[‡]e-mail:gert@pdc.kth.se

In 2001 Eskil Steenberg continued the Verse development by writing a second version of Verse independent of the Institute. In 2002 researchers from the Royal Institute of Technology and the Interactive Institute met to formulate a proposal for a European Commission three-year project formed by seven European research institutes and companies. The project was granted funding and was launched in February 2003 under the name Uni-Verse[2]. The idea with this project is mainly to develop and evaluate 3D tools based on the Verse protocol.

Verse is, at its core, a custom network protocol optimized to describe 3D graphics data according to a certain data format. The data format, in turn, is designed to be flexible and dynamic. Around these two components, we are trying to build an entire platform. By basing the platform on an openly available communications protocol, we try to encourage and simplify independent development and interoperability.

1.1 General Philosophy

The general idea with Verse is to create a platform that can support various kinds of applications involving networked 3D graphics. We want the platform itself to have a fairly "low profile," so that it leaves as many design and policy issues as possible to the application developer. Data describing graphics should be of the highest possible quality, and should describe the objects without regard to how they are rendered by clients. Verse is not intended to be just a research prototype, but should be good enough for real-world use.

2 Architecture

The bulk of this paper will describe various aspects of Verse's architecture, including network and data storage issues.

[1]See http://verse.blender.org/

[2]See http://www.uni-verse.org/

2.1 Client/Server

The Verse protocol is designed to support any network architecture but is often used as a client/server system. Normally there is a single central server, to which multiple clients connect. The server stores the data describing the world, and clients can then connect to the server and read and write the data it stores. The server "knows" which clients are reading which data, and distributes any changes accordingly.

We favor a client/server approach rather than a peer-to-peer or hybrid network architecture for several reasons. One is that by making all the data describing a world reside in a single process, administration and ownership of the world becomes easier to understand. Also, we feel that access security is easier to achieve if there is a single point through which all accesses must go. Persistence also comes naturally in a client/server system: as long as the server is kept running, the world persists.

2.1.1 A Lightweight Server

The most important characteristic of the server in the Verse architecture is that it is small, both theoretically (it has few responsibilities) and in practice (as a computer program it is not very big[3]). Conceptually, all the server does is store data, provide an interface through which that data can be accessed, keep track of which client is accessing what data, and forward changes.

2.1.2 Heterogeneous Clients and Servlets

The word "client" is heavily used when discussing Verse. This is because since the server does so little, much of the work other systems might put in the server is delegated into clients. Programs that are "server-like" in their nature, but technically Verse clients, are often called "servlets". We assume that the administrator of a server also will chose a set of desired additional services and run the required servlets on (or near) the machine that hosts the server itself.

2.2 Data Organization

From one perspective, Verse can be seen as a network-accessible special-purpose database. This section describes how data are stored and managed in Verse.

2.2.1 Nodes

Data stored by the Verse server are divided into a set of discrete entities called nodes. Several distinct types of nodes exist, each one is specialized to store a subset of the data required to describe a world. Currently, the types object, geometry, material, bitmap, text, curve and audio have been defined.

Object nodes are used to represent entities that should be visible in the world. An object is given a "look" by linking it to geometry and material nodes. Material nodes in turn link to bitmap nodes for textures and other image-like data. Text nodes are used to store text, for whatever purpose. Curve nodes store multi-dimensional interpolation curves, for animation. Audio nodes store and stream sampled audio.

Each node instance is identified by a 32-bit unsigned integer number, assigned by the server upon creation. Node IDs are globally unique in each connection to a server. All nodes also support human-readable textual names. Furthermore, each node supports tags, which are simply named values of various types. Tags are collected into named groups.

2.2.2 Node Commands

Each node type defines its own set of commands that operate on the data stored in the node. Node commands are the only thing sent by the Verse network protocol; all communication between a Verse server and its connected clients is done using node commands. Commands are symmetrical, meaning that the same command is often used both as a request and as a reply. Communication in Verse is mainly client-driven; the server does not send unrequested data to clients.

2.2.3 Subscriptions

The concept of subscription forms the basis of Verse's data distribution design. Clients need to actively tell the server which nodes they are interested in, by subscribing to them. Nodes typically contain smaller parts [4] which are in turn subscribable. A client learns about the subscribable parts of a node by subscribing to something at the next higher level, beginning by subscribing to the node itself.

2.2.4 Dynamic Data

One of the most important aspects of Verse, that differentiates it from many existing systems, is that all data stored and handled by it are fully dynamic. Things such as an objects geometric representation, or a color bitmap for texturing, are transmitted and handled so that very small parts of these data structures are always addressable and thereby changeable at any time. For example, any vertex in a Verse geometry node can be moved at any time, polygons can be created and destroyed at will, bitmaps can be repainted on the fly, and so on.

This dynamic nature makes Verse well suited for interactive networked cooperative applications. The data storage formats and the node commands that express them have been designed so that, from a client's perspective, there really is no difference between the original and changed versions of e.g. a vertex position. They both look *exactly* the same, which makes it easier to write clients to support the dynamic properties of the data model.

2.3 Network Layer

Networking is of course important in a system such as Verse. We use a custom-built asynchronous protocol designed to send node commands, layered on top of standard unicast UDP.

2.3.1 UDP

Verse uses UDP, a low-level datagram transport protocol that is part of the TCP/IP standard suite of protocols. UDP, unlike TCP, does not guarantee that datagrams sent actually reach their destination;

---

[3]Currently, the server is roughly 169 KB in binary executable form (stripped, Linux x86)

[4]Such as tag groups, layers, buffers, streams etc.

they can be dropped at any point in the network. This means that Verse must handle dropped datagrams itself, by doing resends.

Verse uses UDP rather than TCP because it is inherently better suited for interactive applications, and also because it gives us more control over the network traffic.

2.3.2 Unicast

Verse datagrams are sent using "classic" unicast semantics, i.e. each datagram has only one recipient. This is in contrast to the use of multicast[Deering 1989], where each datagram is sent to an entire group of recipients. There are many reasons why we do not use multicast in Verse. One is that the basic service provided by it, efficient one-to-many data distribution, does not fit well with a general 3D world. All clients do not want all data; each client only wants the data it is subscribing to.

An alternative might be to use one multicast group per node, but that is not very appealing either. First, IPv4 reserves no more than 24 bits (0.39%) of its address space for multicast groups, while Verse's node ID space is a full 32 bits. Second, separating transmissions into distinct multicast groups means more datagrams in total, which increases the cost of per-datagram overhead. Third, Internet-wide support for multicast is still rather limited.

2.3.3 Asynchronous

The network layer is asynchronous. This means that messages are generally sent one way only, and the sender does not wait for a response before continuing and sending the next message. Messages generally consist of *commands*, which are either system-level or sent to a specific node instance.

Commands are collected into datagrams which are then emitted. The layer decodes the datagram and generates a stream of command invocations, which is delivered to the application-level software. The application might be either the server or a client. Each datagram is handled separately, without regard for the datagrams that preceded it, or the ones that will succeed it. The datagrams are given a sequential number, so that lost (dropped) ones can be detected. When this happens, a resend is eventually done.

When commands are collected into datagrams to be sent, the network layer uses knowledge it has about each command's content to overwrite duplicates when possible. This is a form of *event compression*, and conserves bandwidth by not sending redundant data over the network.

2.3.4 The Verse API

Programmers who wish to develop for Verse do not need to know much about how Verse looks "on the wire". Instead, they use our application programming interface (API), which is delivered as a C link library[5]. By calling functions in the Verse API, a client program can establish a connection to a server, and also exchange data with it. For the most part, functions in the API map 1:1 to node commands, which are sent to the server. When a reply comes back, the client program is notified through a callback function, which is called with the command's parameters (if any) as its arguments. The server is also implemented on top of the same API.

---

[5]Called "libverse.a"

2.4 Object Nodes

Nodes of type object are arguably the most important nodes used in Verse. For something to be visible in a world, it has to be associated in some way with an object node. Each connected client is given an object node that represents that client in the world hosted by the server. This object is known as the client's avatar, but there is nothing special about it. It is just like all other object nodes.

2.4.1 Transform

Verse currently uses a fairly simple transform system to express object movements. It is based on a clock which is synchronized between client and server when the client connects, using a simplistic measurement of the network latency that separates the two. An object's transform consists of the three quantities position, rotation and scale. Changes to each of these three quantities can be done at the zero'th, first or second derivative.

Because all commands that handle transforms using this system include a timestamp for when they should take effect, it is possible to build up a queue of events in advance, if the events are known beforehand. This decouples the network traffic that describes a series of transform changes from the changes themselves, which is sometimes useful.

Setting a transform quantity's value directly (at derivative level 0) is akin to "teleporting" the object, and will likely be restricted in worlds that try to appear realistic. In this case, movement must be done by setting either a velocity or an acceleration. The object-level transform system handles only the entire object as a rigid whole. The animation system supports defining a hierarchical skeleton of bones, with weighted influences on the vertices of the object. Objects can have child objects, to build hierarchical transformations at the object-level, too.

2.4.2 Methods

In addition to the tag system, which stores application-defined "passive" data in objects, there is also support for storing something called methods[Brink 2000]. A method is simply a description of a program entry point associated with the object. The description does not include any code or other implementation of the method; this is left to dedicated clients. Methods can accept parameters of various types, much like functions in C. However, object methods do not have return values, mainly because the underlying network layer is inherently asynchronous.

Methods are collected into method groups, which are named and subscribable. The intended use is to allow various kinds of specialized behavior and/or "intelligence" to be associated with objects. In the full-fledged version of method use, the code that implements each method is stored on the server in a text node, and interpreted by a general virtual machine client. It is, however, entirely possible to write dedicated clients that only deal with handling calls to methods in a given group, bypassing the text node and online code interpretation completely.

As an example of how methods could be used, consider a general 3D world where clients connect and navigate around through the use of avatars. Navigation might be done through the use of keyboard and mouse input, as is common on the PC platform.

Without methods, the browsing client would directly translate keypresses into transform commands sent to the object node representing the avatar. If the avatar is given a set of methods for move-

ment, the client would instead issue calls to these methods when corresponding keyboard keys are pressed. Somewhere, perhaps on the same machine as the server, another client would receive the method calls and translate them into object transform commands. Using the method system in this way allows important properties of an object, such as its control interface, to be abstracted out in a way we find very flexible.

2.4.3   Linking

Object nodes do not contain data about things such as geometry or material properties directly. Instead, such data is stored and managed by instances of dedicated node types, and the object node simply links to these instances as needed. Other nodes can also contain node pointers, for example material nodes often need to refer to bitmap nodes that act as data sources for texture mapping and filtering.

Data stored in a node can be shared between multiple users by simply letting each user link to the desired node. For example, two objects with the same geometry will link to the same geometry node. They can still have different materials, by linking to distinct material nodes.

2.5   Graphics

Since Verse is ultimately a system for building networked 3D graphics applications, it naturally contains quite a bit of mechanism for handling the actual graphics. Specifically, it has a flexible geometric primitive and a material description system.

2.5.1   Creased Subdivision Surfaces

Unlike many other systems and file formats for virtual reality and networked graphics, Verse has a single primitive used to represent graphics. We use a hybrid of Catmull-Clark[Catmull and Clark 1978] and Loop[Loop 1987] subdivision surfaces, and support meshes that mix triangles and quadrilaterals freely. Also, these surfaces have been extended with crease data for vertices and edges, allowing sharp features to be expressed simply and accurately.

2.5.2   Material System

Verse features a very powerful and flexible material system. Rather than supporting a set of pre-defined complete shading models (such as a Blinn shader, a Phong shader, and so on), Verse lets the user define the material from scratch. This is done by building a shading tree, consisting of various fragments. Each fragment represents a simple operation, such as the application of a constant color, the light hitting the surface, blending two colors, and so on. Currently, there are 16 such fragments defined.

3   Working with Verse

3.1   Replacing File Transfers

In the Verse way of working, the 3D data is stored in a server and clients connect to that server and subscribe to the data that each
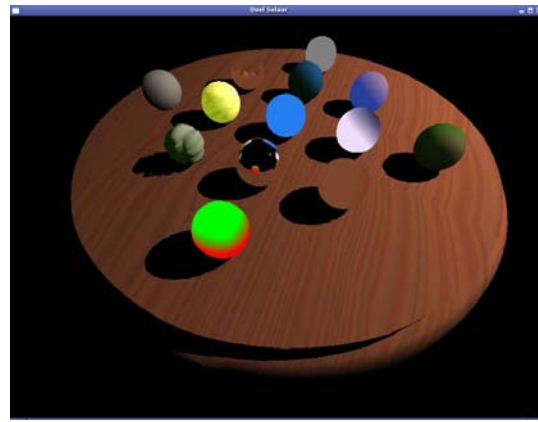

Figure 2: A number of sample materials, including transparency, displacement mapping, reflections, multi-texturing, and refraction.

client is interested in. This means that different tools can collaborate directly and without file transfers and file conversion that often are quite cumbersome for 3D data.

3.2   Single-User Use

Verse is a big advantage also for a single user on a single computer. As a matter of fact several tools can collaborate with Verse as if they were part of a single 3D application, and this is valid on a single computer or distributed over the LAN in a company or distributed over Internet.

Traditionally, it was convenient to use a large monolithic 3D application for all steps in the 3D content creation process. With Verse-enabled tools, it is possible to choose the most appropriate tool for each task, and combine them into a virtual production pipe-line. This also means that tools can be smaller, and much simpler compared to the monolithic approach. This opens the possibility for smaller companies and open source volunteers to implement tools which are optimal for a single task.

3.3   3D Texture Painting

3D texture painting is a simple example of how Verse can speed up the process of content creation. If you are using open source tools you may use The GIMP[Kimball and Mattis 1996-2006] for texture painting and Blender[Blender Foundation 2002-2006] for the geometric modeling. Although Blender has its own set of texture painting tools built-in, being able to directly benefit from the much larger set of tools available in applications such as The GIMP would be a boon to artists. Without Verse, artists that want to use The GIMP to manipulate textures need to go through a save/reload cycle. With Verse the artists can see the result in the 3D world directly as they paint

3.4   Distributed Use

Today's companies and organizations become more and more geographically distributed, and the 3D content creation industry is no exception. Just having a customer review with the customer on a remote location and the possibility to change the model interactively

and have the customer review and directly comment on the changes opens up new ways of working.

3.5 Available Clients

In the Uni-Verse project a number of new Verse clients have been developed, and popular existing 3D tools have been adopted to communicate using Verse.

A set of tools which has been developed for Verse from the beginning consists of the modeling tool *Loq Airou*, the symbolic data manipulation and inspection tool *Connector*, the UV tool *UV Edit* and the geometry layer painter *Layer Painter*.

A tool called *Purple*[Brink 2005] for manipulation of Verse data with a data flow graph approach has been developed, hoping to illustrate how a future 3D package might be designed around Verse, and also to help get programmers "on board", by lowering the threshold to program for Verse.

Several rendering clients exist. *Quel Solaar* (see Figure 1) is high-quality, high-performance rendering client which has been designed for Verse from the beginning. This means that Quel Solaar supports changing the rendered data at any time. Quel Solaar also supports global illumination, dynamic shadows, the OpenGL 2.0 shader language, reflections, refractions and transparency as well as displacement mapping.

Furthermore, a rendering client based on OpenSG has been developed. It supports clustered rendering, where many computers cooperate to render for a single physical display.

A Verse plug-in to *Autodesk 3ds Max*[Autodesk Corporation 2006] has been developed, as well as a Verse connection for the major open source 3D package *Blender*.

Finally, an acoustic simulation client is being developed. The client makes realistic simulations of a building model in Verse, for use by architects and acoustic consultants. The model can be changed interactively and the acoustic simulation is then automatically updated. To achieve interactive speeds of the acoustic simulation the complexity of the geometric model is first automatically reduced (by a separate, dedicated client).

4 Conclusions

Verse is a network protocol for dynamic exchange of 3D data in a variety of situations. Verse clients have been developed for the standard tasks in 3D modeling and rendering.

The Verse way of connecting applications has been shown to increase productivity in the content creation process. We have the goal to establish Verse as an open industry standard for connecting 3D applications interactively.

5 Acknowledgments

References

AUTODESK CORPORATION, 2006. 3ds max. `http://www.autodesk.com/3dsmax`.

BLENDER FOUNDATION, 2002-2006. Blender. `http://www.blender.org/`.

BRINK, E., AND STEENBERG, E., 2006. The Verse Specification. `http://verse.blender.org/cms/fileadmin/verse/spec/`.

BRINK, E. 2000. *Dynamic Method Calls In A Networked 3D Environment (TRITA-NA-E0077)*. Master's thesis, Royal Institute of Technology.

BRINK, E., 2005. Purple. `http://purple.blender.org/`.

CATMULL, E., AND CLARK, J. 1978. Recursively generated b-spline surfaces on arbitrary topological surfaces. *Computer Aided Design 10*, 350–355.

DEERING, S. E. 1989. Host extensions for ip multicasting. RFC 1112, IETF, November.

KIMBALL, S., AND MATTIS, P., 1996-2006. The GNU Image Manipulation Program. `http://www.gimp.org/`.

LOOP, C. 1987. *Smooth subdivision surfaces based on triangles* Master's thesis, Department of Mathematics, University of Utah, Salt Lake City.