Special Effects and Rendering

Proceedings from SIGRAD 2002
held at
Linköpings universitet, Norrköping, Sweden,
November 28th and 29th , 2002
organized by
Svenska Föreningen för Grafisk Databehandling
and
Norrköping Visualization and Interaction Studio

Edited by

Mark Ollila

Published for Svenska Föreningen för Grafisk
Databehandling (SIGRAD) by
Linköping University Electronic Press
Linköping, Sweden, 2002

LINKÖPING UNIVERSITY
ELECTRONIC PRESS

Linköping Electronic Conference Proceedings, No. 7
Linköping University Electronic Press
Linköping, Sweden, 2002

ISBN 91-7373-464-0 (print)
ISSN 1650-3686 (print)
http://www.ep.liu.se/ecp/007/
ISSN 1650-3740 (online)

Print: UniTryck, Linköping, 2002

Table of Contents

Welcome to SIGRAD2002 in Norrköping!

SIGRAD had a big year in 2002, as we officially signed a cooperation agreement with ACM SIGGRAPH. This agreement means that ACM SIGGRAPH and SIGRAD will work to promote the field of computer graphics and interactive techniques through cooperation, collaboration and the free exchange of information.

The goal of the SIGRAD2002 conference is to provide a Nordic (yet international) forum for presenting recent research results in computer graphics, with a focus on special effects and rendering, and to bring together experts for a fruitful exchange of ideas and discussion on future challenges.

This year is the first time that the SIGRAD conference has spanned two days. Over the past 20 years we have had a single day of invited presentations from experts in academia and industry. To help foster the development of a computer graphics research community in Sweden (and also the Nordic) it was decided to have an actual Call-for-Papers to help form a focused research day, and hence, a two day event.

The nuclei of the research day are examples of research and work in progress from all over Sweden. We also have research presentations from Scotland and Finland, an important step in building a critical mass of local and regional cooperation for future projects in the area of computer graphics. The second day, can be seen as a traditional SIGRAD, with invited keynote presentations from leading experts from both academia and industry. These include experts from the USA, Spain, Finland, and internal to Sweden in disciplines such as gaming and special effects. The day begins with a keynote presentation from Dr. Paul Debevec, from the Graphics Lab at the Institute of Creative Technology. We would like to thank our many experts for accepting our invitation to present at SIGRAD2002.

We are very pleased with the good response to the call for papers, which resulted in 17 submissions, of which 9 were accepted for publication through a blind review. We hereby wish to thank the program committee and contributing reviewers for putting together a strong program, which, by spanning from image based rendering, to augmented reality allows us to have a broad coverage of the field. The conference is supported by the SIGRAD, Norrköping Municipality and NVIS. Thanks should go to Linköping University Electronic Press for arranging the electronic publishing of this conference. Last but not least, special thanks are due to the organizing committee for making the conference possible.

We would like to express our gratitude and warm welcome to the keynote speakers, authors of contributed papers, and other participants. We wish you a most pleasant stay in Norrköping.

Mark Ollila
Chair, Program committee

Anders Ynnerman
Chair, SIGRAD

SIGRAD2002 Program Committee

SIGRAD2002 consisted of experts in the field of computer graphics from all over Sweden. We thank them for their comments and reviews.

SIGRAD2002 Organizing Committee

SIGRAD Board for 2002

The Light Stage: Photorealistically Integrating Real Actors into Virtual Environments

Paul Debevec
USC Institute for Creative Technologies
www.debevec.org

The key to achieving realism in much of visual effects is to successfully combine a variety of different elements - matte paintings, locations, live-action actors, real and digital sets, CG characters and objects - into a single shot that looks like it was all there at the same time. An important, subtle, and frustratingly complex aspect of this problem is to match the lighting amongst these elements. Not only do the objects and environments need to be lit with the same sources of light, they need to properly reflect and shadow each other. In our graphics research, we have explored ways of using advanced lighting simulation techniques and philosophies to integrate CG objects, digital characters, and live-action performances into real and synthetic environments with physically correct and perceptually believable lighting.

In our 1999 film Fiat Lux, we used the Facade photogrammetric modeling system to model and render the interior of St. Peter's Basilica from a set of high-dynamic range digital photographs. The film called for this space to be augmented with numerous animated computer-generated spheres and monoliths. The key to making the computer-generated objects appear to be truly present in the scene was to illuminate the CG objects with the actual illumination from the Basilica. To record the illumination we used a high dynamic photography method we had developed in which a series of pictures taken with differing exposures are combined into a radiance image -- without the technique, cameras do not have nearly the range of brightness values to accurately record the full range of illumination in the real world. We then used image-based lighting to illuminate the CG objects with the images of real light using the RADIANCE global illumination rendering system, also calculating the cast shadows and reflections in the floor. The full animation may be seen at: http://www.debevec.org/FiatLux/

Most movies star people rather than spheres and monoliths, so much of our research since then has focused on the more complex problem of realistically rendering people into real and virtual scenes. In this work we have designed a series of Light Stage devices to make it possible to light real people with light from virtual sets. Version 1 of the Light Stage was designed to move a small spotlight around a person's head (or a small object) so that it is illuminated from all possible directions in about a minute - the amount of time a person can comfortably stay still in a neutral expression. It consisted of a two-bar rotation mechanism which can rotate the light in a spherical spiral about the subject. During this time, a set of stationary digital video cameras record the person or objects' appearance as the light moves around, and for some of our models we precede the lighting run with a geometry capture process using structured light from a video projector. From this data, we can then simulate the object's appearance under any complex lighting condition by taking linear combinations of the color channels of the images in the light stage dataset. In particular, the illumination can be chosen to be measurements of illumination in the real world or the illumination from a virtual environment, allowing the image of a real person to be photorealistically composited into such a scene with correct illumination. Light Stage 1 may be seen at: http://www.debevec.org/Research/LS/

Light Stage 2 was designed to shorten the amount of time needed to acquire a light stage dataset so that a person could be captured in a variety of natural facial expressions. This set of expressions could then be used as morphing targets to produce an animated version of the person. Light Stage 2 consists of a semicircular arm three meters in diameter that rotates about a vertical axis through its endpoints. Attached to the arm are twenty-seven evenly spaced xenon strobe lights, which fire sequentially at up to 200 Hz as the arm rotates around the subject. The arm position and strobe lights are computer-controlled allowing the strobes to synchronize with high-speed video cameras.

In 2001 we applied the Light Stage 2 capture process to capture a number of Native American cultural artifacts including an otter fur headband, a feathered headdress, an animal-skin drum, and several pieces of neckwear and clothing. We were able to show these artifacts illuminated by several real-world natural lighting environments, and designed a software program for interactively re-illuminating artifacts in real time. Images from this project can be seen at: http://www.debevec.org/Research/LS2/

Light Stage 1 and 2 provided the necessary proof-of-concept to build Light Stage 3, which can achieve realistic composites between a actor's live-action performance and a background environment by directly illuminating the actor with a reproduction of the direct and indirect light of the environment into which they will be composited. Light Stage 3 consists of a sphere of one hundred and fifty-six inward-pointing computer-controlled light sources that illuminate an actor standing in the center. Each light source contains red, green, and blue light emitting diodes (LEDs) that produce a wide gamut of colors and intensities of illumination. We drive the device with measurements or simulations of the background environment's illumination, and acquire a color image sequence of the actor as illuminated by the desired environment. To create the composite, we implemented an infrared matting system to form the final moving composite of the actor over the background. When successful, the person appears to actually be within the environment, exhibiting the appropriate colors, highlights, and shadows for their new environment. Images and videos from Light Stage 3 can be seen at: http://www.debevec.org/Research/LS3/

This talk will present joint work with Tim Hawkins, Andreas Wenger, C.J. Taylor, Jitendra Malik, HP Duiker, Westley Sarokin, Dan Maas, Mark Sagar, Jamie Waese, Andrew Gardner, and Chris Tchou.

Reconciling theory with the realities of production
Michael Conelly, Lighting Supervisor, Rhythm and Hues

Michael Conelly is a Senior Lighting Supervisor for Rhythm and Hues in the USA. He creates new toolsets for lighters, supervise senior technical directors and manages productions. His recent work has been seen in "Scooby Doo" and he is currently working on "Cat in the Hat" and "X-Men 2".

State of the Art in the Nordic Special Effects Industry address
Viktor Björk, Co-founder of Swiss AB

Viktor Björk has been involved in the Visual Effects-industry in Sweden for over seven years. As of today he works as CEO at the newly founded company Swiss that has recently finished character animation and compositing of a new Moby promo for the track 'In This World'. Viktor Björk will give an overview of the status of the Special Effects-industry in the Nordic region.

The King is Dead, Long live the King - The new emerging Production Pipeline
Peter Zetterberg, Founder of UDS

Peter Zetterberg is the founder of UDS. A company that has produced games for many platforms such as the PS, PS2, PC etc. Peter will discuss the production pipeline and how the Special FX community and the Gaming community are merging closer together.

Distribution of Film Digitally - The HUB
Mats Erixon, Advanced Media Technology Lab, KTH

Mats Erixon is involved in building up the digital HUB for distribution of film content to digital cinemas across Sweden. He has a deep interest in uncompressed distribution of moving images and over 30 years experience in the film industry. Mats Erixon, Advanced Media Technology Lab,

Global Illumination Rendering - the Arnold way
Marcos Fajardo, Spain

Marcos is the creator of the Arnold Global Illumination rendering package. Recently, he has been working on some shots for the movie The Core which involved rendering 18 million bubbles with inter-reflections, shadows and extreme motion blur.

The future of Mobile Graphics and Rendering
Jani Vaarala, Research Scientist, Nokia Mobile Phones

Jani Vaarala is a research scientist at Nokia Mobile Phones in Finland. He is currently working on implementation of mobile graphics architectures for rendering and augmented reality.

Implementation of a Dynamic Image-Based Rendering System

Niklas Bakos[1], Claes Järvman[2] and Mark Ollila[3]

Norrköping Visualization and Interaction Studio
Linköping University

Abstract

Work in dynamic image based rendering has been presented by Kanade et al. [4] and Matusik et al. [5] previously. We present an alternative implementation that allows us to have a very inexpensive process of creating dynamic image-based renderings of digitally recorded photo realistic, real-life objects. Together with computer vision algorithms, the image-based objects are visualized using the *Relief Texture Mapping* algorithm presented by Oliveira et al [6]. As the relief engine requires depth information for all Texels representing the recorded object in an arbitrary view, a recording solution making depth extraction possible is required. Our eyes use binocular vision to produce disparities in depth, which also is the most effortless technique of producing stereovision. By using two digital video cameras, the dynamic object is recorded in stereo in different views to cover its whole volume. As the depth information from all views are generated, the different views from the image-based object are textured on a pre-defined bounding box and relief textured into a three dimensional representation by applying the known depth disparities.

1 System Prototype

The first step in the process is to record a dynamic object in stereo, which gives us the photo textures for the image-based object and the possibility to derive depth information from the stereo image-pairs. To be able to use the recorded video as a texture when rendering, it is important that one camera (i.e. the left) is installed parallel to the normal of the sides of the bounding box surrounding the object, and the other (i.e. the right) next to, in a circular path so that both cameras have the same radius to the object. As we are interested in the recorded object only, the image background should be as simple as possible. By using a blue or green screen, the object can easily be extracted later on. A blue screen can easily be installed by using cheap blue matte fabric on the walls. Depending on the amount of cameras available, the dynamic object is recorded in stereo in up to five views (front, back, left, right and top). In this project, only two cameras were used, giving us only one view when filming the dynamic object. As the recording is finished, the video streams are sent via firewire to a PC, where the resolution is rescaled to 256x256 pixels, the background is removed and the depth maps are calculated, enhanced, cropped and sent to the relief rendering engine. (Pipeline in figure 1).

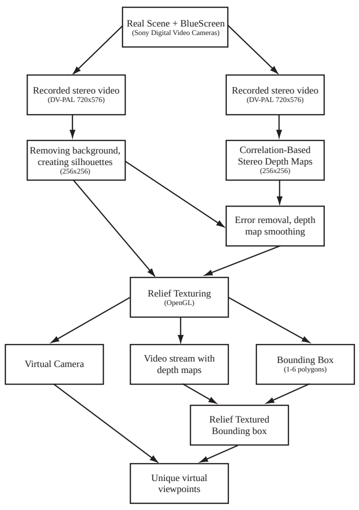

Real Scene + BlueScreen
(Sony Digital Video Cameras)
Recorded stereo video
(DV-PAL 720x576)
Recorded stereo video
(DV-PAL 720x576)
Removing background, creating silhouettes
(256x256)
Correlation-Based Stereo Depth Maps
(256x256)
Error removal, depth map smoothing
Relief Texturing
(OpenGL)
Virtual Camera
Video stream with depth maps
Bounding Box
(1-6 polygons)
Relief Textured Bounding box
Unique virtual viewpoints

Figure 1: Prototype overview. *A schematic view over the different stages required in the process of rendering new views of an image-based object.*

2 Depth approximation

When the stereo video have been recorded and streamed to the computer client, our algorithms start processing the data to create useful video frames and information about the scene. As the objects are extracted from the original video, the process of estimating the depth of the scene is initiated. When the approximated depth map for a certain frame is generated, it is used together with the object image to render unique views, using the relief-rendering engine. This session starts with a brief overview of the depth algorithm, followed by complete descriptions about all the steps from using original video streams to sending a finalized depth map and video frame to the rendering process of virtually viewing the object from an arbitrary view.

[1] nikba@itn.liu.se
[2] claja622@student.liu.se
[3] marol@itn.liu.se

2.1 Algorithm overview

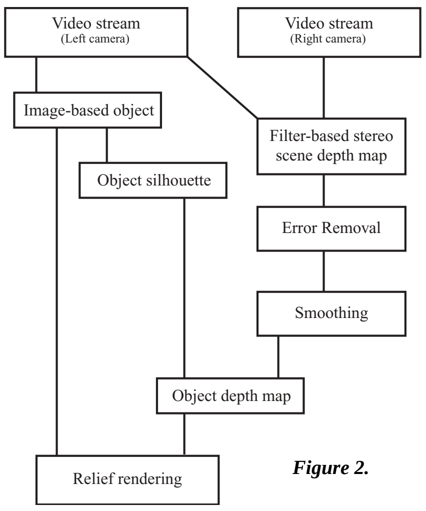


Figure 2.

A summary of the algorithm pipeline is shown in figure 2. From the *N* stereo video cameras, we have *2N* video streams. From the left camera (which sees the scene straight from the front), the object-only video frames and silhouette will be created. As the scene is recorded with a blue screen background, both the silhouette and the object extraction are created rapidly. Simultaneously, both the left and the right video streams are segmented into frames and sent into our filter-based depth algorithm. At this stage, the frames can be downsized for optimization purpose, which will result in faster depth map approximations with lower quality. For each frame, each pixel from the left image is analyzed and compared with a certain area of the right image to find the pixel correspondence. With this known, the depth could be estimated for each frame. Since this mathematical method outputs a relatively distorted image, it needs to be retouched to fit the relief engine better. First, the depth map is sent to an algorithm for detecting edges, where an edge could be thought of as noise, distorting the depth map, and removed by pasting the intensity value of neighboring pixels. With the errors removed, the depth approximation of the image-based object will contain less noise and unnecessary holes, but disparities between contiguous object regions might be rendered with too sharp intensity variances, which will exaggerate the displacement of some object parts when applying the relief mapping. To solve this, the depth map is smoothened and finally, the silhouette is added to remove approximated background depth elements.

2.1.1 Filter-based stereo correspondence

The method implemented in our system prototype uses filter-based stereo correspondence developed by *Jones* and *Malik* [2], a technique using a set of linear filters tuned in different rotations and scales to enhance the features of the input image-pair for better correlation opportunities. A benefit of using spatial filters is that they preserve the information between the edges inside an image. The bank of filters is convolved with the left and the right image to create a response vector at a given point that characterizes the local structure of the image patch. Using this information, the correspondence problem can be solved by searching for pixels in the other image where the response vector is maximally similar. The reason for using a set of linear filters at various orientations is to obtain rich and highly specific image features suitable for stereo matching, with fewer chances of running into false matches. The set of filters F_i (fig. 3) used to create the depth map consists of rotated copies of filters generated by

$$G^{\chi}_{n,0}(x,y) \,|\, G_n(u)\Delta G_0(v) \quad ; \quad u \,|\, x\cos\chi\, 4\, y\sin\chi \,, \; v \,|\, x\sin\chi\, 4\, y\cos\chi$$

where $n=1, 2, 3$ and G_n is the n^{th} derivative of the Gaussian function, defined as

$$G_0(x) \,|\, \frac{1}{\sqrt{2\phi\omega^2}} e^{\frac{4z^2}{2}} \quad ; \quad z \,|\, \frac{x}{\omega} \quad G_1(x) \,|\, 4\frac{1}{\omega} z G_0 \quad ;$$

$$G_2(x) \,|\, \frac{1}{\omega^2}(z^2\, 4\, 1)G_0 \quad ; \quad G_3(x) \,|\, 4\frac{1}{\omega^3}(z^3\, 4\, 3z)G_0 \,.$$

The matching process was performed using different filter sizes to find the optimized filter settings, resulting in an *11x11*-sized matrix with a standard deviation value ω of *2*. The number of filters used depends on the required output quality. Using all filters would result in a high detailed depth approximation, but the processing time would be immense. Testing different filters to optimize speed and output quality, the resulting filters consisted of nine linear filters at equal scale, with some of them rotated, as shown below.

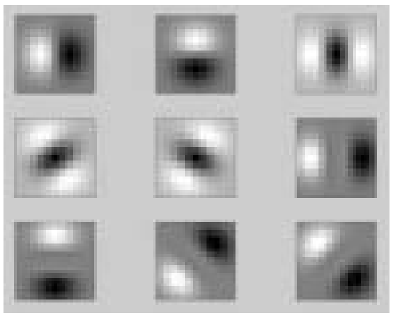


Figure 3: Spatial filter bank. *Image plots of the nine filters generated by copies of rotations of Gaussians.*

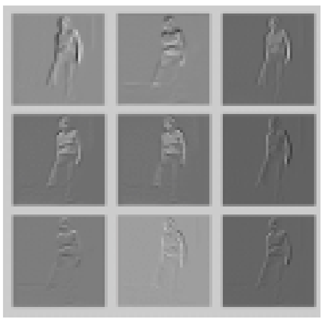


Figure 4: Response vectors. *An illustration of how the response vectors will look like after being convolved with different filters. In reality, a response vector never represents a whole image.*

The disadvantage of using one scaling level only is the loss of accuracy when matching pixels near object boundaries or at occluded regions. But again, using more scales, the rendering time will increase proportionally. To search for pixel correspondence, an iterative process is created, scanning the left image horizontally, pixel by pixel, left to right, and seeks for similar intensity values inside a defined region surrounding the current pixel location. For each row, the set of linear filters are convolved with a region of the right image determined by its width and the height of the filter size, to create a response vector that characterizes the features of this row. At this row, a new response vector for each pixel is created by convolving the filter bank with a filter-sized region from the left image. How the convolved response vectors for a whole image would look like is illustrated in figure 4. (Note that the response vectors are only representing a small region of the image for each iteration of the correspondence process).

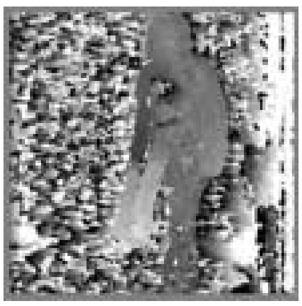
$$v_{i,right} = \text{Right image } (r) \otimes F_i = \sum_{x'}\sum_{y'} r(x', y') F_i(x - x', y - y')$$

$$v_{i,left} = \text{Left image } (l) \otimes F_i = \sum_{x'}\sum_{y'} l(x', y') F_i(x - x', y - y')$$

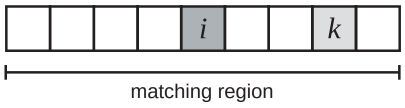
The convolving returns only those parts of the convolution that are computed without the zero-padded edges, which minimizes the response vectors and optimizes the whole process of finding the correspondence. As soon as the images are convolved with the filters, the matching process for finding the correlation is initiated. To restrict the searching area, a one-dimensional region needs to be determined. By using a small region, the corresponding pixels may not be found, as the equivalent pixel probably is located outside this region. On the other hand, if the region is too large, a pixel not related to that area might be thought of as correct. When the region is established, this is used to crop the response vector $v_{i,right}$ created from the right image. When the response vectors are defined at a given point, they need to be compared in some way to be able to extract some information about how the pixels are related. By calculating the length of their vector difference *e*, which will equal zero if the response vectors are identical, this can be used to solve the correspondence problem. This is done by taking the sum of the squared differences (SSD) of the response vectors,

$$e = \sum_{i} \left(v_{i,left} - v_{i,right}\right)^2$$

Figure 5.



where *i* is the amount of filters used and the pixel position (defined as *k*) containing the value closest to zero is saved. When the correspondence has been established, the disparity has to be defined to be able to create a depth map. For each pixel in the left image, we know the position of the matching pixel in the right image. To create a connection between this data, the depth value $d(i, j)$ for each pixel could be estimated by



$$d(i, j) = k - i$$

where *k* is the horizontal position of the corresponding pixel and i is the current pixel position. The depth map (fig. 5) is approximated with intensity levels depending on the size of the constant defining the size of the matching region and if a corresponding pixel is found to the left of current pixel *i*, the intensity is set to a value pointed to white, and vice versa, depending on the rotation of the image-pair.

2.1.2 Locating errors and noise

The primary depth map image generated by the filter-based stereo algorithm is a general approximation of the depth information regarding the objects in the video frames. As this algorithm has no knowledge in form of estimating the structure of object connectivity or how the scene is designed, unpredicted outputs might appear. They can be found by convolving the image with an edge detection filter [7]. The operator best suited for our needs turned out to be the *Robinson* filters h_1 and h_3.

$$h_1 | \begin{pmatrix} 1 & 1 & 1 \\ 1 & 42 & 1 \\ 41 & 41 & 41 \end{pmatrix}$$

$$h_3 | \begin{pmatrix} 41 & 1 & 1 \\ 41 & 42 & 1 \\ 41 & 1 & 1 \end{pmatrix}$$

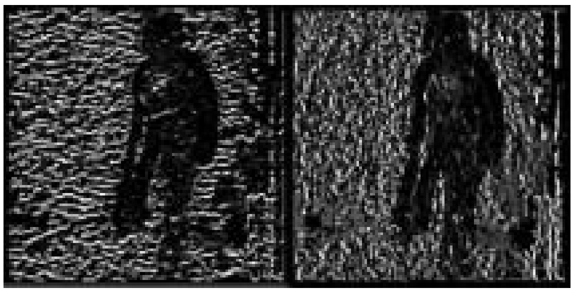


Figure 6.

With the vertical and the horizontal Robinson filters defined, they are convolved with the depth map to find obvious edges in it, using the convolution formula for two dimensions. We now have two temporary depth map images, with the edges defined vertically and horizontally, shown in figure 6. From this, the edge magnitude of each pixel could be derived as

$$d(x,y) \,|\, |d_1(x,y)| \,2\, |d_2(x,y)| \,|\, \sqrt{d_1{}^2(x,y) \,2\, d_2{}^2(x,y)}$$

The result is shown in figure 7a and gives a better analysis of how the errors are structured. To be able to use this information cleverly, the pixels convolved and defined as positions of eventual errors need to be saved. Also, these pixels need to be easily accessed. By using a threshold value, we can decide which of the convolved 'edge'-pixels that will belong to the error pixels in the original depth map, shown in figure 7b. With the positions of the erroneous pixels known, they are replaced by neighboring pixel values, which creates a smoother depth map, although not mathematically perfect, since it is only assumed that these pixels have the same properties as the invalid and replaced ones. On the other hand, the noiseless depth maps, shown in figure 8, will generate tremendously enhanced renderings when applied by the relief engine.

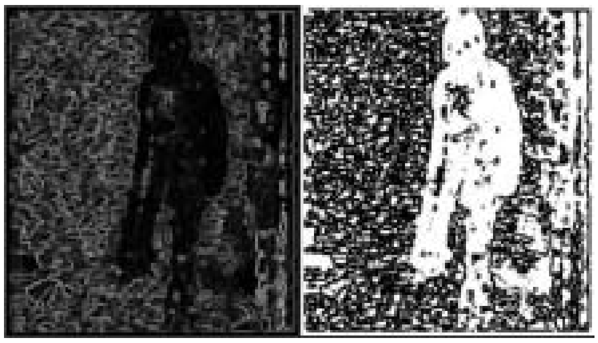


Figure 7a & b.

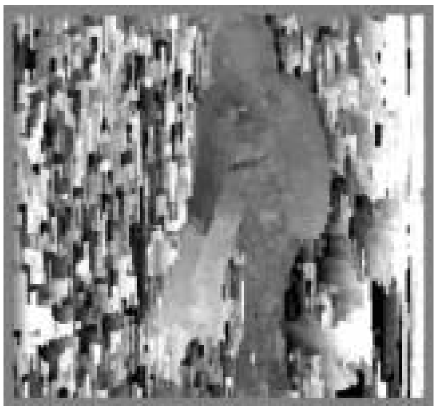
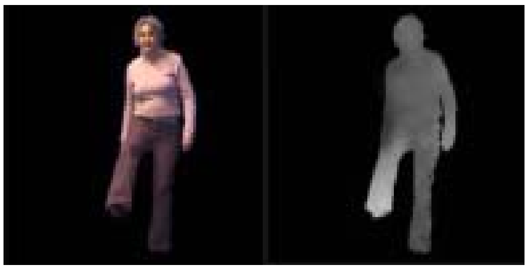


Figure 8.

Figure 9.

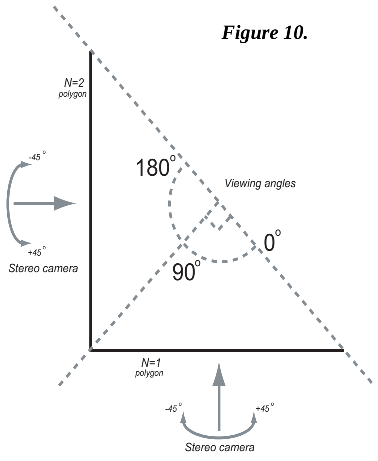


2.1.3 Smoothing the depth map

The output from the edge detection process is a more or less error free depth map, regarding the hole filling and the depth intensity interpretation. On the subject of intensity, it can fluctuate significantly over connected and contiguous surfaces over the object. As some intensity values diverges in areas were they actually would be similar, the solution would be to decrease the higher values and increase the lower to create more similar intensities over that specific area, in other words, smoothing the image. This might generate an intensity value incorrect for the true depth of that part of the object, but applying this solution to the whole image, the displacement would act as an intensity threshold only. The Gauss function is used to generate a smooth depth map, defined as the well-known *Gaussian blur* filter [1]. We defined a Gaussian operator and convolved it with the depth map to obtain the smooth result, seen in figure 9.

2.1.4 Rendering

Figure 10.



A fully functional application for the relief rendering of the image-based object and its depth maps was written in C++ using OpenGL, created in parallel to this project [3] and modified to fulfill the criterion of our system prototype. The number of polygons required for rendering equals the amount of stereo cameras used. Because of the good depth information approximated with the filter-based stereo algorithm, the viewing angle was set to $\partial 45 N^{\text{Â}}$ from the center of the origin of the textured polygon box, illustrated in figure 10.

3 Results

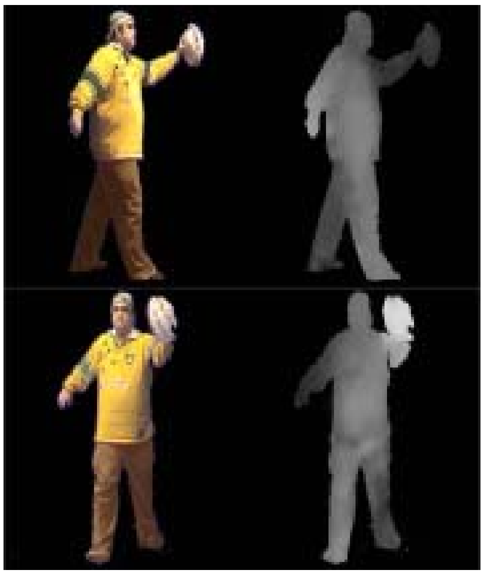
The resulting application consists of two demos (screenshots available on the last page):

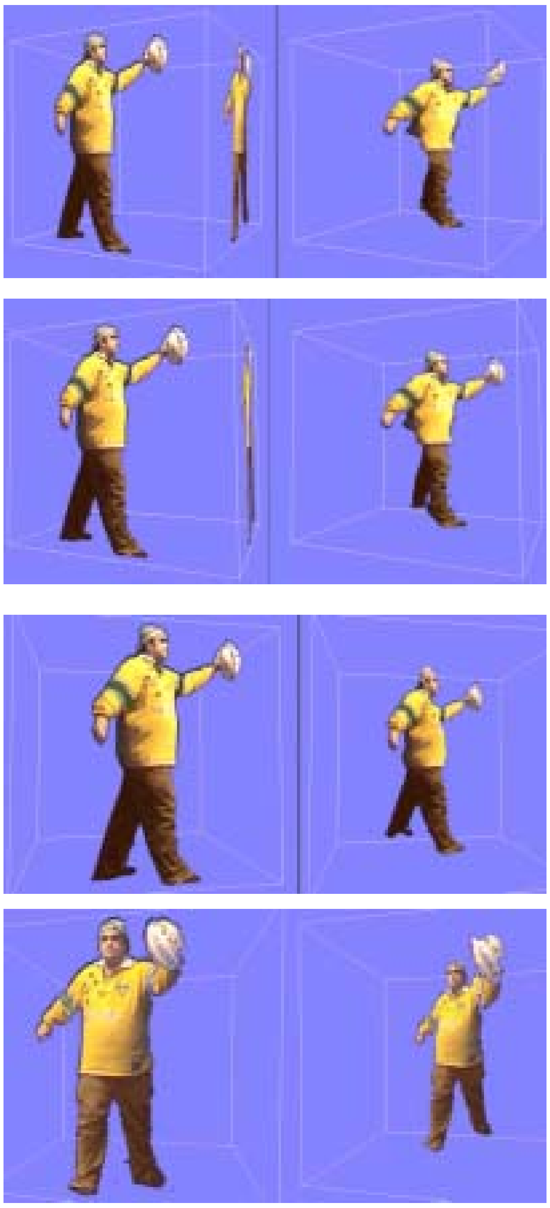
∉ Static demo (yellow pullover) - Requires two input textures and with two depth maps, textured on two polygons. From two original views, with 90 degrees separation, new unique views can be created within 180 degrees. The polygons are mapped with textures of size 256x256 pixels and the frame rate is ~15 frames/sec.

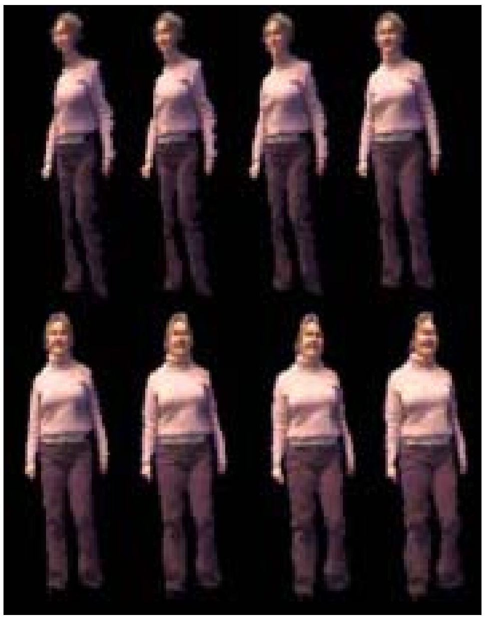
∉ Dynamic demo (pink pullover) - Representing a person walking around. Textured on only one polygon, which restricts the viewing angle to 90 degrees. The amount of input data required depends on the frame rate. We used a frame rate of 20 frames/sec, with a video buffer of 40 images and 40 depth maps. The relief engine had no problems with rendering a constantly updating image buffer and the animated sequence showed no indications of flickering.

References

[1] BOGACHEV, V. 1998. Guassian measures. *Mathematical Surveys and Monographs 62*.
[2] JONES, D., AND MALIK, J. 1992. ”A computational framework for determining stereo correspondence from a set of linear spatial features”. In *EECV*, 395–410.
[3] JÄRVMAN, C., “Static and Dynamic Image-Based Applications using Relief Texture Mapping”, Linköping University, LITH-ITN-MT-20-SE. May 2002.
[4] KANADE, T., NARAYAN, P., AND RANDER, P. W. 1997. Virtualized reality: Constructing virtual worlds from real scenes. *IEEE Multimedia 4*, 1, 34–47.
[5] MATUSIK, W., BUEHLER, C., RASKAR, R., GORTLER, S. J., AND MCMILLAN, L. 2000. Image-based visual hulls. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., 369–374.
[6] OLIVEIRA, M. M., BISHOP, G., AND MCALLISTER, D. 2000. Relief texture mapping. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., 359–368.
[7] SONKA, M., HLAVAC, V., AND BOYLE, R. 1996. *Image Processing, Analysis, and Machine Vision*, second ed. Brooks/Cole Publishing Company.







SIGRAD (2002)
Mark Ollila (Editors)

Snow Accumulation in Real-Time

Håkan Haglund,[1] Mattias Andersson[2] and Anders Hast[3]

University of Gävle
Department of mathematics, nature and computer science
[1]na99hhd@student.hig.se
[2] mattias.andersson@gavle.to

[3] Creative Media Lab
aht@hig.se

Abstract
Whenever real-time snowfall is animated in computer games, no snow accumulation is simulated, as far as we know. Instead, so-called zero thickness is used, which means that the blanket of snow does not grow when the snowflakes reach the ground. In this paper we present a method for simulation of snow accumulation, which simulates the different stages, starting with a snow free environment and ending with a totally snow covered scene, all in real-time. The main focus is not on the physical properties of snow but on speed and visual result.

1. Introduction

Snow is one of the most complex natural phenomenon. It has the ability to transform a rocky landscape to a soft cotton like blanket in only a few hours. One of the most fascinating properties of snow, is that it is not constant but changes form and appearance from day to day, depending on factors like wind and temperature.

To reproduce the snows properties in computer graphics is an challenging task. There exist a few fine examples where realistic snow environments has been created, but so far no one has done a realistic snowfall with accumulation in real-time. When snow occur in computer games it has zero-thickness. In this paper we present a method for simulating snow accumulation. The focus is neither on the physical properties of snow, nor on how the snow falls through the sky. Instead it is on visual result and the possibility of using the proposed algorithm in real-time.

2. Previous Work

Law et al.[7] describes a method for simulating how snow accumulates over alp terrain. The work is concentrated on visualizing a realistic blanket of snow from long distance.

Summers et al.[10] deals with simulation of how sand, mud and snow deforms. To be able to create a deformable surface, the surface is divided into rectangular voxels with different height values. When an object touches the surface, the surface is deformed and the material is moved to surrounding voxels.

Fearings[3] algorithm generates very nice and realistic snow. He divides his algorithm into two parts and together they generate a thick blanket of snow on the ground. The first part is the accumulation model. It decides how much snow every surface will get considering flake flutter, the influence of wind and snows ability to get stuck on an uneven vertical surface. The second part of the algorithm handles the stability of the fallen snow. It moves snow from instable places to stable.

Of the methods mentioned above, Fearings is the one producing the most realistic blanket of snow. However the produced images has taken hours to render. Furthermore, none of the mentioned methods are suitable for real-time rendering.

3. The Model

The main idea is to use a two dimensional matrix in order to store information about snow depth over certain areas where snow might fall. How detailed the blanket of snow would be is determined by the size of the matrix and the size of the accumulation area. It would be preferable to have a matrix where each cell is in the size of a single snowflake. This would yield a very nice and detailed rendering of the snow blanket. However, since a real-time simulation is the goal, each cell must correspond to a much larger area. Thus, we have a trade off between speed and visual appearance. Nonetheless, this is no big problem and the rendering turns out to be visually plausible.

The height information was the used for making a triangulation over the area in question. This was then rendered as the a cover of snow. In the beginning of the snow fall when just a little amount of snow have reached the ground, the snow cover is more or less transparent. Therefore, blending was used in order to imitate this impression.

The snow fall itself was animated by using a particle system, where each particle correspond to a single snowflake. During the snowfall, the individual flakes will finally reach the snow cover and then the corresponding cell in the matrix is updated. Furthermore, the particle animating the flake will itself be destroyed.

Each snowflake was modeled by using billboards[11]. They are two dimensional images which always are oriented towards the camera. A polygon model of snowflakes is simply too expensive to use and billboards turn out to give a convincing impression of real snow fall.

3.1. Triangulation

Triangulation of scattered points can be done in many ways[9]. In this case the points are uniformly distributed and the triangulation is easily implemented. When the matrix is traversed, two triangles is created using the height information in four neighboring cells in the matrix. As shown in figure 1 the triangulation could be done in two different ways.

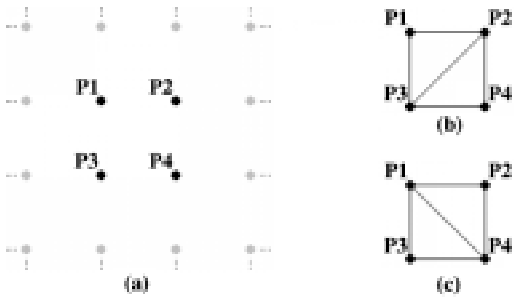


Figure 1: *(a) Current points in the matrix. (b,c) Possible triangulations.*

The natural thing would be to choose one orientation and use it for the triangulation of the whole matrix as shown in figure 2(a). However, it turns out that the way the triangulation is done affects the resulting image negatively, since vertical lines will appear in the image as shown in figure 2(b).

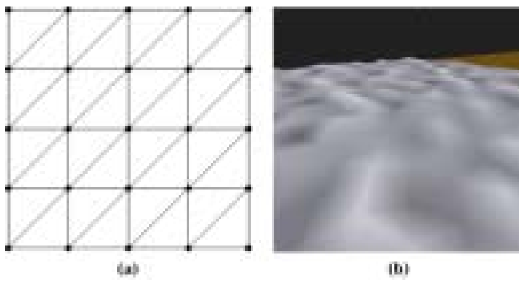


Figure 2: *(a) A triangulation pattern, where all triangles are oriented the same way. (b) Diagonal lines are visible.*

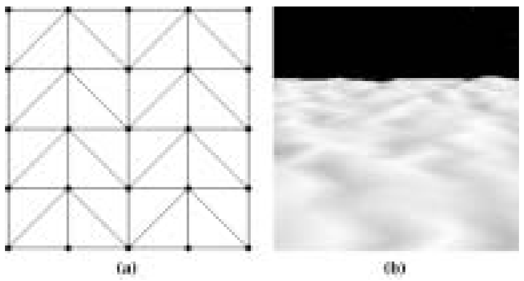


Figure 3: *(a) Triangulation pattern, where every other triangle is oriented in the opposite direction. (b) A zigzag pattern becomes visible.*

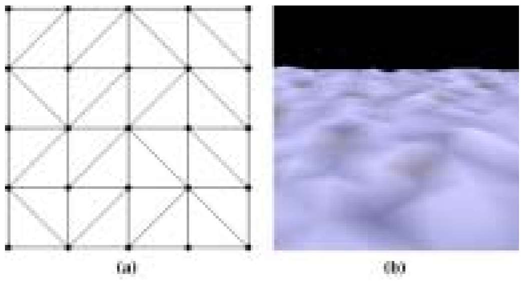


Figure 4: *(a) A triangulation pattern which seams to be more random. (b) No repeating pattern shows in greater extent.*

Another triangulation scheme was used in order to get rid of the visible lines and it is shown in figure 3(a). This resulted in a zigzag pattern instead, which is visible in figure 3(b). Therefore a more random triangulation was made, similar to the way Cook[1] handles textures, to avoid repeating patterns. The first two triangle pairs on a row was oriented one way and the next two oriented the other way. Furthermore, every other row was displaced to get irregularities vertically, as shown in figure 4(a). Zigzag patterns can come up but only occasionally and only in smaller sizes, which can be seen in figure 4(b). These patterns could probably be by dimished using some stochastic scheme. Moreover, this would probably cost more computationally and since our goal was to implement a real-time simulation, the third triangulation pattern was used in our model.

3.2. Shading

Since the triangles are rather large due to the speed criteria, the triangles must be shaded using Gouraud[4] shading or even Phong[8] shading. Flat shading would make the blanket of snow look very angular and sharp. Since OpenGL was used, Gouraud shading was chosen for the animation.

To be able to do proper shading, each normal for every points in the triangulation had to be computed. This could be done in several ways. Either to compute the normalized average of all triangles sharing that vertex, as proposed by Gouraud[4]. Another and eventually faster method would be to make an approximation. Since speed is crucial for a real-time rendering, we chose to make a fast approximation.

Every point has got four closest neighboring points in the four main directions. Hence, it is possible to obtain four gradient vectors. The vector to the right and the vector downwards was used to obtain one normal by computing the cross product. The vector to the left and upwards was used in order to compute a second normal. The average of these normals was then used as the normal of that particular vertex. Two special cases had to be treated differently. At the corners and at the edges, there is not four neighboring points available. Then, only one normal was computed from two gradient vectors, to be used in the shading computation.

It could be tempting to compute only one normal from two neighboring points for the whole triangulation, since it will be even faster. For flat surfaces the difference was actually very small. However, when the difference in height was larger, the result was not acceptable. Especially surfaces with sharp edges as the ridge on the roof, like on the left house in figure 10, was not shaded in a convincing way. The normals will point straight upwards on the ridge with the more advanced method. This made the snow look very soft over the ridge. However, with the method only considering three points, the normals at the ridge will point in the same direction as the other normals on one roof side. This gives a peculiar shading effect where the roof ridge seems to be skewed towards one roof side.

3.3. Blending

Whenever a snowflake intersects the surface the closest height value is increased. As explained earlier, this will affect a much bigger area of the snow surface than the size of a small snowflake, due to the trade off between speed and accuracy. The result is that a large area will go from the state of having no snow to the state of having snow, after the first snowflake reaches that area. This will clearly not yield a convincing simulation. Because the surface should not go from no snow at all, to suddenly be a solid white surface, the snow had to gradually tone in. For this purpose blending was used. How transparent the snow should be was decided in every point by the snow depth in that point. When the depth was zero the snow was totally transparent. How thick the snow should be in order to be completely white with no transparency, was decided differently for each type of material that the snow where about to cover.

The blending factor was decided in every point of the matrix and linearly interpolated over the triangles. Thus, every corner of the triangles could have different blending factors. One triangle could be totally transparent in one corner and white in another. Figure 7 and 8 shows how blending is used to give the impression of that the surface starts to having a thin layer of snow. After all, snow is not really opaque. Instead, the snow cover will actually be transparent when it is rather thin. After some more snowing, the blending will turn the triangles completely white. After that the snow cover has no blending and the snow cover continues to increase while the triangles are raised by using the stored height values.

3.4. At the Edge of the Snow Cover

The edges of the snow cover had to be treated differently. This is clearly shown in 6. In this case, not only the surface of the snow cover had to be rendered, but also the sides of the snow had to be triangulated and rendered.

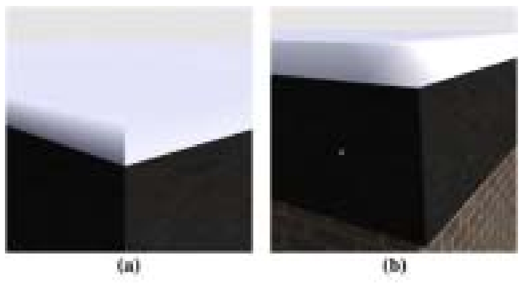


Figure 5: *(a) Corner with no maximum value for the snow depth. (b) Corner with maximum value along the edges.*

The sides were triangulated from the edge of the snow surface and down to the surface beneath. Because the sides are vertical, the normal at the ground was set to be perpendicular from the sides and the normal at the surface was set equal to the normal for that particular point in the matrix. The shading would make an illusion of soft edges, due to the linear interpolation of the light intensity. This illusion only worked satisfactory for thin snow. However, when the snow blanket became large as in figure 5(a) the corner looks very sharp and edgy. Therefore a maximum value for the snow depth was set at the edges. Hence, the edges could not grow as much as the interior of the area. This problem is handled by the stability criterion by Fearing. Anyway, in the real world, snow will fall of the sharp edges, and the snow will not be as high on the edge as in the interior. The result, using maximum height is softer edges, even with rather thick snow as shown in figure 5(b).

4. The Simulation

In the animation the blanket of snow was divided into four sections and thus four matrices. One on each house, one on the road and one on the pavement. At the edge on the road up on the pavement a maximum value was set for the thickness of the snow in order to make th edge softer as explained previously. When the snow on the road eventually reached this value, the maximum value was removed and the two blankets was connected to each other. This gave a smooth and soft blanket of snow over the hard and sharp pavement edge. Moreover, snow tend to get a bit thicker close to a wall. In order to illustrate this, the value of the snow height was simply increased at the corresponding cells.

Figure 6 - 10 are a few snapshots from simulation. The textures in the model was borrowed from Hill [6].

5. Discussion

There are several possible improvements that could be made to the proposed model. Nonetheless, we have kept things simple with the real-time criterion in mind. The rendering of snow is done in such way that the blue part of the RGB color is a bit stronger as it is in nature. However, the snow cover will be rather smooth due to the linear interpolation used in Gouraud shading. A more sofisticated shading model like the Cook Torrance[2] model would probably yield a much more convincing snow like appearance of the snow cover. Nevertheless, it would take much more time to render. Other possible improvements are mentioned in the Future Work section.

As computers becomes faster, it should be possible to use a matrix where each cell corresponds to a smaller area than used in our simulation. Hence, the snow cover could be more detailed and thus yielding a more convincing snow cover. A drawback with using large areas for each cell, is that foot prints could not be done in the snow. However, if smaller areas are used, this is possible. Again, the trade off between speed and visual appearance has to be taken into account for each case.

Even though the normal computation works quite well, it is more or less an good estimation of the normal. A better way to compute the normal is of course by taking all the polygons into account that share the vertex in question. Another way to go about would be to use a spline filter to reconstruct the normal as is done by Hast et al.[5] Since no cross product is necessary in their approach, this could turn out to be a feasible solution. At least if a reconstruction filter of lower degree is used.

6. Conclusions

A new method for snow accumulation was proposed, where speed is crucial without sacrificing visual appearance. A heigh value matrix was used to store current snow depth. These height values where used to triangulate the area, and the triangles were rendered with Gouraud shading which is fast. The combination of blending and triangulation gives the impression of snow slowly accumulating on a surface. First the snow will give the impression of being transparent since the snow cover is thin. After a while the snow cover becomes opaque and it will grow during the simulation. The instability of snow at edges was modeled by using maximum values for these places, giving the impression of smooth snow covered edges.

The model should be easy to implement in real-time 3D games as long as the ground is not cluttered with small objects that must have their own height value matrix.

6.1. Future Work

One important feature that should be implemented in a realistic simulation, is the influence of wind. This is not an easy thing to implement since it will affect stability and also the edges. Furthermore, snow that is affected by wind will accumulate faster near walls etc. An efficient model handling all these cases should be possible to derive.

Another interesting possibility is to use bump-mapping[11]. It is probably not feasible to let each bump represent an snowflake that reaches a specific area. However, precomputed bump maps can be used in order to enhance the visual appearance of the shaded triangles. Which shading model would be preferable for snow should also be ascertained.

References

1. R. L. Cook. Stochastic sampling in computer graphics. In *ACM Transactions on Graphics, vol 5*, pp. 51 - 72, 1986.
2. R. L. Cook and K. E. Torrance. A Reflectance Model for Computer Graphics. In *Computer Graphics*, 15(3), pp. 307-316, 1982.
3. P. Fearing. Computer modeling of fallen snow. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 37 - 46, 2000.
4. H. Gouraud. Continuous Shading of Curved Surfaces, *IEEE transactions on computers* vol. c-20, No 6, June 1971.
5. A. Hast, T. Barrera, E. Bengtsson. Reconstruction Filters for Bump Mapping *WSCG'02*, Poster, pp. 9-12, 2002.
6. P. Hill. http://www.planetunreal.com/hillgiant, 2002-05-10.

7. S. Law, B. M. Oh and J. Zalesky. The synthesis of snow covered terrains. http://www.graphics.lcs.mit.edu/boh/Projects/snowGen-FinalWrite.html, 1996.

8. B. T. Phong, Illumination for Computer Generated Pictures *Communications of the ACM*, Vol. 18, No 6, June 1975.

9. J. O'Rourke. Computational Geometry in C Second Edition. *Cambridge University Press*, 1998.

10. R. W. Summer, J. F. O'brien and J. K. Hodgins. Animating sand, mud and snow. In *Proceedings of Graphics Interface*, pp. 125- 132, 1998.

11. A. Watt. 3D Computer Graphics Third Edition. *Addison-Wesley*, 2000.

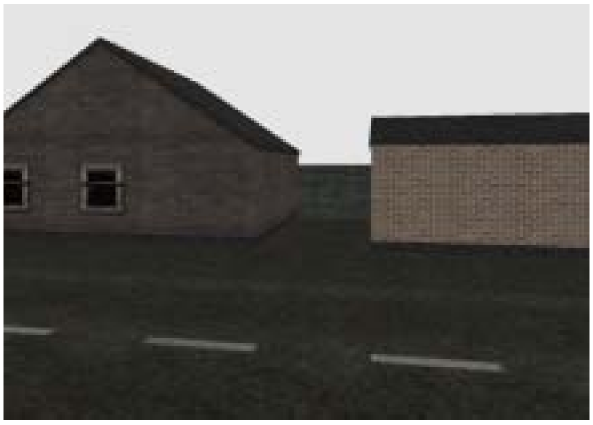


Figure 6: *The model were it shall snow.*

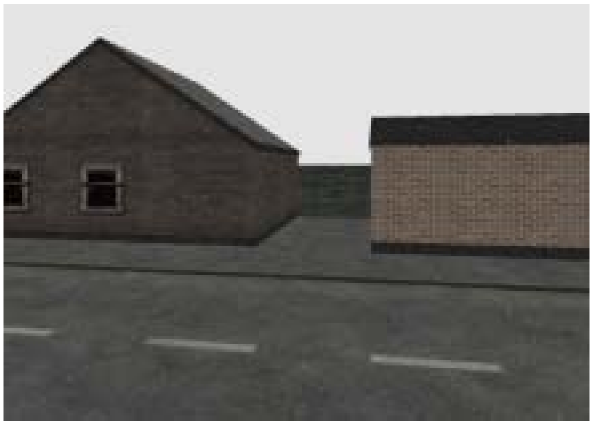


Figure 7: *The model after about 5 minutes of snowing.*

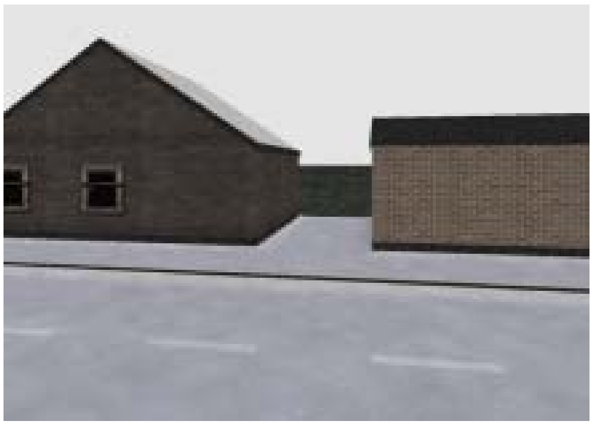


Figure 8: *The model after about 12 minutes of snowing*

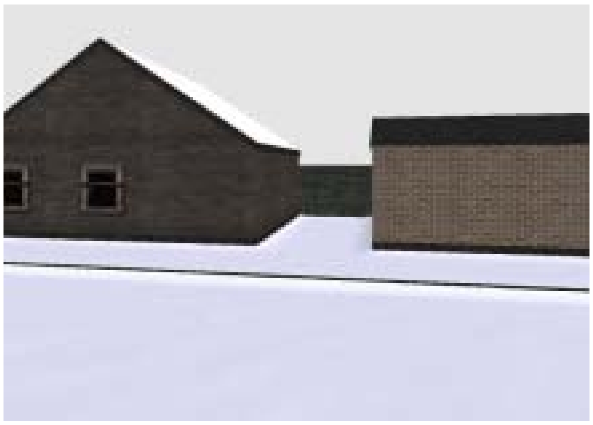


Figure 9: *The model after about 25 minutes of snowing*

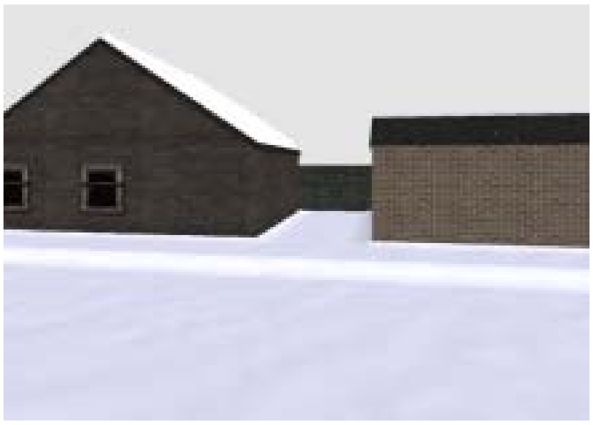


Figure 10: *The model after about 50 minutes of snowing.*

SIGRAD (2002)
Mark Ollila (Editors)

Animation of Water Droplet Flow on Structured Surfaces

Malin Jonsson

University of Gävle,
Kungsbäcksvägen 47, S-801 76 Gävle, Sweden.
na99mjn@student.hig.se

Anders Hast

Creative Media Lab
University of Gävle,
Kungsbäcksvägen 47, S-801 76 Gävle, Sweden.
aht@hig.se

Abstract

Several methods for rendering and modeling water have been made and a few of them address the natural phenomenon of water droplets flow. As far as we know, none of those methods have used bump maps in order to simulate the flow of a droplet on structured surfaces. The normals of the bump map, that describes the geometry of the micro structured surface, are used in the flow computation of the droplets. As a result, the water droplets will meander down on the surface as if it has a micro structure. Existing models were not suitable for this purpose. Therefore, a new model is proposed in this paper. The droplet will also leave a trail, which is produced by changing the background texture on the surface. This method will not present a physically correct simulation of water droplets flow on a structured surface. However, it will produce a physically plausible real-time animation.

1. Introduction

There is an endless ever-changing kingdom of phenomenon provided by the nature that is possible to model, animate and render. These phenomenons offers, with their complexity and richness, a great challenge for every computer artist. Several natural phenomenons, like fire, smoke, snow, clouds, waves, trees and plants, have with different success been modeled in computer graphics through the years. Several different methods that address the problems of rendering and modeling water and other similar fluids have been developed since the 1980t's. Most of them concern animation of motion in water in forms of waves and other connected fluids and surfaces, i.e. whole bodies of water. For example have oceans waves [10] [5] [13] and waves approaching and braking on a beach [12] been modeled. Realistic and practical animation of liquids [2] [3] has also been made. Only a few methods that have been proposed during the 1990's address the problems of the natural phenomenon of water droplets. Methods for simulating the flow of liquids were proposed to render a tear falling down a cheek [4] and changes in appearances due to weathering [1]. Different methods for animation of the flow of water droplets running down a curved surface with [7] or without obstacles on it [8] have also been proposed. Different ways to create droplets have been used [6], for example metaballs that are affected by the gravitation were used as one solution [14]. It is quite difficult to simulate the flow of water droplets for the purpose of high-precision engineering, due to the complicated process that the flow and the shape of the droplet represent. This process has many unknown factors that plays a big role. The shape and the motion of a water droplet on a surface depend on the gravity force that acts on the droplet, the respective surface tensions of the surface and the water droplet, and the inter-facial tension between them[6]. Shape and motion is also under the sway of other things like air resistance and evaporation. These effecting factors can be divided into two different groups. As an example, gravity and wind can be placed in the group of external forces. Factors like surface tension and inter-facial tension belongs to the group of internal forces. To be able to create an accurate physical simulation of the phenomenon of water droplets, a tremendous amount of forces and factors would have to be taken into account. As mentioned above many of the dominant factors for water droplets are still unknown not only within computer graphics but also within physics. To the long list of effecting factors these ones can be added:

- Motion of the water within the droplet.
- The capillarity of the surface.
- The interaction forces between each point on the surface of the droplet and the solid surface.

1.1. Main Contribution

Trying to take all of these different factors into account would create an accuracy that goes far beyond what is possible to do in the scope of this paper. A method is proposed for generating an animation of the flow of water droplets on a structured surface. Instead of creating a structured surface with a huge amount of polygons, a bump mapped[9] flat surface is used. Furthermore, the bump normal is used to control the motion of the droplets. To our knowledge, this has never been investigated before. Hence, the droplet will meander down the surface and move as if it actually was flowing on a structured surface. However, as mentioned earlier, all the different factors which have an influence on water droplets and their flow, have not been taken to account in the method. The aim of this paper is not to make a simulation that is physically correct at every point, but to make a plausible animation of droplets meandering down on a bump mapped surface.

2. Previous Research

There are at least four published papers about droplets and their flow that address similar problems as this paper.

2.1. Animation of Water Droplets on a Glass Plate

Kaneda et al[6] propose a method for realistic animation of water droplets and their streams on a glass plate. The main purpose is to generate a realistic animation, taken into account gravity of water droplets, inter-facial tensions and merging of water. Those are the dominant parameters of dynamical systems. A high-speed rendering is also proposed, which takes reflection and refraction of light into account. Their method will reduce the calculation cost of animations that contains scenes seen through a rainy windshield or windowpane.

The route that the water a droplet takes as it meanders down on a glass plate is determined by impurities on the surface and inside the droplet itself. To be able to animate water droplets and their stream a discrete surface model is developed and the surface of the glass plate is divided into a mesh. Figure 1 shows a lattice that is used on a glass plate. To every lattice point on the glass plate an affinity, 0-1, for water is assigned in advance.

A water droplet begins to meander down a surface when

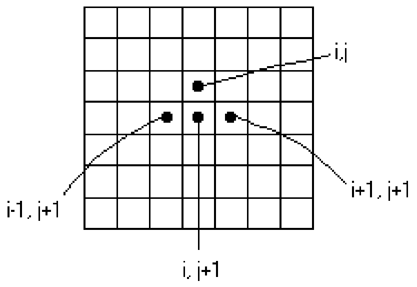


Figure 1: *A discrete surface model, with the droplet at position (i,j)*

the mass exceeds a static critical weight. To simulate the meandering the droplet at point (i, j) can move to one of three different points on the lattice, as shown in Figure 1. If some water exists on any of the three points, the droplet will move to the lattice point with the direction $(i, j+1)$ has the highest priority.In case there is no water already existing on the different points, a value depending on for example the angle of inclination is used as a decision parameter. They claim that the speed of the droplet is not depending on the mass of the droplet. Instead it depends on the wetness and the angle of inclination of the glass plate. When two droplets collides and merges the speed of the new droplet is calculated by using equation law of conservation of the momentum. A meandering droplet that has no water ahead will decelerate and when the dynamic critical weight is larger than the mass of the droplet, it will finally stop.

2.2. Animation of Water Droplet Flow on Curved Surfaces

The previously proposed method is not able to simulate a water droplet on a curved surface, which is an important and necessary technique for drive simulators. Therefore an extended method for generating realistic animation of water droplets and their streams on curved surfaces is proposed by Kaneda et al [8]. The dynamics, such as gravity and inter-facial tension that acts on water droplets is also taken into account in this method. Two different rendering methods that takes refraction and reflection into account, is also proposed. One method pursues photo-reality with help of a high quality rendering. The other proposes a fast rendering method that uses a simple model of water droplets.

A discrete surface model is used to make it possible to simulate the flow of droplets running down the curved surface. The curved surface is divided into small quadrilateral meshes and may be specified by Beziér patches. It is converted to a discrete model, using a quadrilateral mesh with a normal vector at the center. Affinity contributes to the meander of the streams and to the wetting phenomenon. The degree of affinity for water is assigned to each mesh in advance.

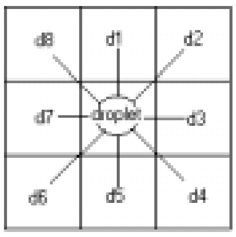


Figure 2: *The eight directions of movement*

This value describes the lack of uniformity on a surface, for example a glass plate. The uniformity can be impurities and small scratches.

The droplet is affected by gravity and wind. When these forces exceed a static critical force, the water droplet starts to meander down the surface. The critical force originates from the inter-facial tension between water and a surface and is the resistance that prevents the droplet from moving. The direction of movement is classified into eight different directions as shown in figure 2. The probabilities for each direction is calculated based on three different factors. The first one is the direction of movement under circumstances in which it obeys Newton's law of motion. The second factor is the degree of affinity for water on the meshes next to the droplet. The last one is the wet or dry condition of the eight neighboring meshes. The water droplet is moved to the next mesh when the direction of movement is determined and if the accumulated time exceeds a frame time, the droplet is moved to the next mesh.

A solution to the wetting phenomenon that appears when a droplet meander down a surface, as well as the problem with two droplets merging, is also addressed. Two different methods for rendering water droplets are proposed. The fast version use spheres. The more sophisticated use meta-balls.

2.3. Simulating the flow of liquid droplets

Fournier et al [4] present a model that is oriented towards an efficient and visually satisfying simulation. It focuses on the simulation of large liquid droplets as they travel down a surface. The aim is to simulate the visual contour and shape of water droplets when it is affected by the underlying surface and other force fields.

The surface is defined as a mesh of triangles. At the beginning of the simulation a "neighborhood" graph is built. In this graph each triangle is linked to the triangles adjacent to itself. Through the entire simulation each triangle knows which droplets are over it as well as every droplet know which triangle it lies on at the moment. Adhesion and roughness is considered in this method. The adhesion is a force that works along the surface normal. A droplet will fall from a leaning surface if the adhesion force of the droplet becomes smaller than the component of the droplets acceleration force that is normal to the surface. The roughness of the surface is assumed to only reduce the tangential force.

The motion of droplets is generated by a particle system, where droplet is represented by one particle each. This representation offers many advantages for simulations that have a wide spectrum of behaviors, because of the generality and flexibility such systems can offer. A droplet might travel over several triangles between two time steps. To ensure that the droplet is properly affected by the deformations on the surface it has traversed, the motion of the droplet over each individual triangle is computed. When a droplet travel from one triangle to another, the neighborhood graph is used to quickly identify which triangle the droplet moves to. The two forces gravity, and friction, which affects the water droplets, are assumed to be constant over a triangle.

2.4. Animation of Water Droplets Moving Down a Surface

Kaneda at al[7] propose a method for generating an animation with water droplets that meander down a transparent surface. A large amount of droplets are used to generate a realistic and useful animation for drive simulators. There method employs a particle system in which water droplets travel on a discrete surface model. The proposed method involves extensions of previously discussed papers[6,8]. One of the main achievements is modeling of obstacles that act against water droplets, like the wiper on the windshield.

The curved surface is divided into small quadrilateral meshes and the droplets move from one mesh point to another under the influence of external forces and obstacles. The degree of affinity for water is assigned in advance to each mesh. Affinity describes the lack of uniformity on an object surface due to such things as small scratches and other impurities. The degree of affinity in most cases is assigned randomly based on a normal distribution in order render the droplets meandering and wetting phenomenon.

By taking into account some dominant factors the direction of movement can be determined. The dominant factors that affects the meandering of water droplets that is mentioned the paper is:

1. Direction of movement under circumstances in which it obeys Newton's law of motion.
2. Degree of affinity for water of the neighboring meshes.
3. The wet or dry condition of the neighboring meshes
4. Existence of obstacles on the neighboring meshes

A stochastic approach is taken for determining the direction of movement, because the route of the stream cannot be calculated deterministically. This is due to the many unknown factors that play a role. This means in other words that the direction of movement is classified into eight different directions, as done in an earlier mentioned paper [8]. The probabilities of movement for every direction is calculated with the four dominate factors, described above, taken into account.

The method for rendering water droplets which is proposed in this paper is based on a method that is published by Kaneda et al [6]. The method uses environment mapping to generate realistic images of water droplets. Spheres are used to approximate the water droplets. The contact angle of the water on the surface is taken into account. This method has been extended further in this paper. Such factors as defocus and blur effects are added to generate more realistic images.

3. Droplet Flow Controlled by Bump maps

The different factors that have an affect on the flow of the water droplet are almost countless. Hence, a correct animation is more or less impossible to make. The goal of this paper is therefore to make a physically plausible animation that will produce a natural looking animation of the flow. A real wetting effect which will affect other droplets was not be implemented. Neither was a method for merging of droplets. A simple solid sphere was used to model the droplets. An animation was implemented using C++ and OpenGL. In the animation a flat surface is modeled using a texture and a bump map which is retrieved from the texture. An object oriented particle system was used where each droplet is a particle. This will make the animation easy to control. Furthermore, it is easy to add more droplets to the animation.

3.1. External and internal forces

There are different forces that acts on the water droplets as they meander on the surface. The different forces can be divided into two groups, the external forces, $\mathbf{f}^{ext}$, and the internal forces, $\mathbf{f}^{int}$. Kaneda et al[8] set the external forces to be gravity and wind. However, we will set the external force to be gravity only, since no wind is applied in the proposed model. Nonetheless wind or any other external force could be added if applicable. Moreover, we will use the same denotation of vectors as used by Kaneda et al and also introduce some new vectors.

The internal force is a force of resistance and its direction is opposite to the direction of movement, $\mathbf{d}_p$:

$$\mathbf{f}^{int} = -\alpha \mathbf{d}_p. \tag{1}$$

The resistance originates from the inter-facial tension that exists between the water droplet and the surface. The affinity which is denoted α is in advance experimentally set to some value, which is assumed to be constant all over the surface for simplicity.

3.2. Direction of movement

The direction of movement can be computed by applying the Gram Schmidt orthogonalization algorithm[11] as shown

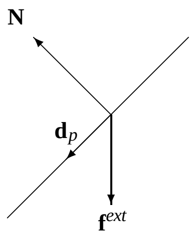


Figure 3: *The direction of movement d_p for a bump with normal* **N** *and gravity* $\mathbf{f}^{ext}$

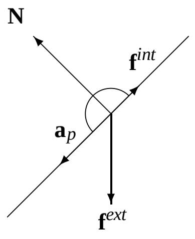


Figure 4: *Forces acting on the droplet*

in figure 3:

$$\mathbf{d}_p = \mathbf{f}^{ext} - \left(\mathbf{N}\cdot\mathbf{f}^{ext}\right)\mathbf{N}. \tag{2}$$

The normal vector **N** is the unit length normal which is retrieved at every point from the bump map. This normal will affect the water droplets as they meander down the surface. It will appear as the droplets are directed in a natural way by the visual bumps on the surface underneath the droplet. Furthermore, the whole polygon has a main direction downwards or tangent **T**, computed from the external force $\mathbf{f}^{ext}$ and the normal of the polygon $\mathbf{N}'$:

$$\mathbf{T} = \mathbf{f}^{ext} - \left(\mathbf{N}'\cdot\mathbf{f}^{ext}\right)\mathbf{N}'. \tag{3}$$

The bi-normal of the plane is computed as:

$$\mathbf{B} = \mathbf{T}\times\mathbf{N}'. \tag{4}$$

In order to calculate the acceleration of the water droplet, the mass, m, and the forces that acts on the droplet, $\mathbf{f}^{ext}$ and $\mathbf{f}^{int}$, are used. The acceleration $\mathbf{a}_p$ shown in figure 4 is then decomposed into the component toward the direction of movement $\mathbf{d}_p$, by projecting it onto this vector[8]:

$$\mathbf{a}_p = \frac{\left(\mathbf{f}^{ext}+\mathbf{f}^{int}\right)\cdot\mathbf{d}_p}{m}\mathbf{d}_p. \tag{5}$$

The velocity **v** of the droplet is computed by adding the acceleration $\mathbf{a}_p$ to the velocity for each step. Similarly, the velocity is added to the position P. Furthermore, the velocity

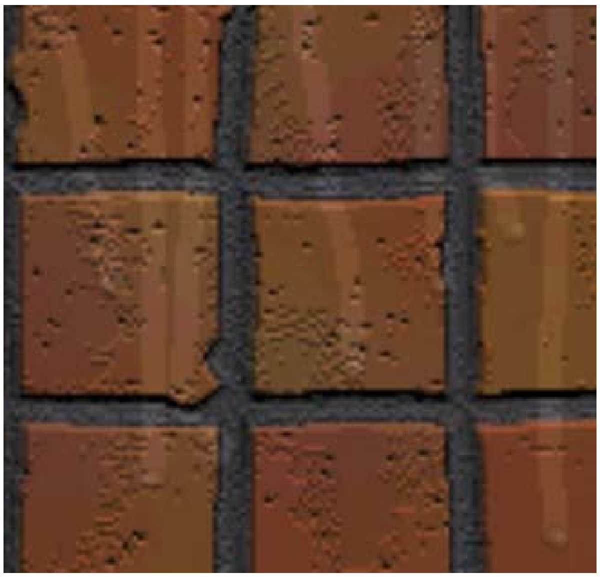


Figure 5: *One frame from the droplet animation. The trails show that the droplets are affected by the underlying bump mapped surface.*

must be projected down onto the plane, in order to prevent the drop from leaving the surface, which of course is modeled in nature by other forces. Nevertheless, this will work for our purposes. This algorithm gives us the new position of the droplet and the droplet is moved to that point during one frame of animation. Hence, the following computations are necessary besides computing the acceleration:

$$\mathbf{v}_{i+1} = v_i + \mathbf{a}_p, \tag{6}$$

$$\mathbf{v}_p = \mathbf{T}(\mathbf{T} \cdot \mathbf{v}_{i+1}) + \mathbf{B}(\mathbf{B} \cdot \mathbf{v}_{i+1}), \tag{7}$$

$$P_{i+1} = P_i + \mathbf{v}_p. \tag{8}$$

3.3. Speed Control

In nature, water meandering down a surface will not accelerate up to full speed, due to several of the forces mentioned in the introduction. Therefore, a speed controller was implemented, delimiting the speed in two ways. First a maximum speed was introduced. Secondly, the speed will be reduced on bumpy areas. Thus, letting the droplet flow rapidly on flat surfaces, but be slowed down considerably on bumpy areas. A bumpy area is defined as a position where:

$$\mathbf{N} \cdot \mathbf{N}' < 1 - \varepsilon, \tag{9}$$

where ε is a threshold value that can be used to control how large the bumps should be in order to slow down the droplet more than usual bumpiness would.

3.3.1. The trail

In order to produce a natural looking trail on the surface which the droplet has traversed, a texture map is used. A snapshot from the animation is shown in figure 5 In the animation a texture map is used and the height map is derived from it. The trail is produced by altering the glossiness of the part of the texture that the droplet has passed. It can easily be confirmed by looking on a wall, on which water have been poured on, that the thin layer of water in the trail reflects light with a higher degree of specularity than the underlying surface has.

4. Discussion

The aim is to make an animation that look as natural and realistic as possible. Because of that and several physical factors that still are unknown for the flow of water droplets, there are lot of tampering that needs to be done with the different parameters. The only way to get a satisfying result is to experiment with the different values and see what is going to happen.

As shown earlier the velocity is projected down to the plain in order to prevent the droplet from leaving the surface. This is something that maybe can be controlled in a smoother way. For example, a factor that makes the droplet adhere to the surface would be one way to handle it.

The wetting effects that the flow of a droplet has on the traversed surface is only implemented as a change of the specular light in the trail after the water droplet. If the wetting phenomenon were to be more correct implemented, the droplet would for instance leave a small amount of water behind as it flows down the surface. This would reduce the size of the droplet and finally make it stop. The wetting phenomenon would also make other droplets that comes near a trail of water adhere to it. Subsequently it would flow almost strictly in the same trail as the droplet before. The problem with two merging droplets is not addressed in this paper.

Only one normal is retrieved from the map for each droplet. Nonetheless, it is also possible to use several normals for this computation. The droplet is after all covering more than one position in the height map. It turns out not to be an good idea to compute an average of the normals involved to use in the droplet computations, since the effect on the droplet will be diminished due to the averaging.

Another way to use more than one normal would be to define the bumpiness which should slow down the droplet as described earlier. If the average deviation of the normals from the mean normal, under the droplet, is larger than some threshold value, then the area is considered being bumpy.

Even though the animation is realistic, nature can sometimes surprise you. This is especially true for droplets on a structured surface. Sometimes droplets will not meander straight down on a totally flat surface. Instead they will meander sideways. By making an experiment where two pictures are taken, one of the structured surface and one of a water droplet that flow over the surface, a comparison of the simulated result and the real thing could be done. The picture taken on the surface would subsequently be used as a texture and a bump map could be retrieved from it in order to produce an animation. The result would show how far from the real thing the animation is.

The proposed model in will make the simulation of the flow of a water droplet on a structured surface considerable faster than if polygons were used to form the micro structure. The object itself will also be rendered faster.

5. Conclusions

A method for animation of water droplets flow on structured surfaces was proposed and the droplets in this method were affected by the underlying bump mapped surface. The proposed method will save time due to the use of a bump mapped surface instead of a larger amount of different triangles. Several parameters where used for the animation of the flow of a water droplets, like the gravity working on the droplet, affinity of the surface, and the mass of the droplet used in the proposed method. Moreover, is the algorithm simplified in such way that adhesion to the surface on bumpy areas is modeled by slowing down the drop. A maximum speed is also used for modeling adhesion. All this will make the animation of the individual droplets fast.

5.1. Future Work

There are several improvements that can be done to the proposed method. Moreover, there are various of extensions that can be employed to the present method. Some examples of possible improvements are, to make a better simulation of the wetting phenomenon, so that it will affect the droplets size and shape. Hence, droplets will become smaller as they leave a trail.

Another improvement would be to create the droplet with help of Beziér curves and let the control points be altered by the bump mapped normals. Different normals should affect the different parts of the droplet, making the droplet stretch and bend. The shape of the water droplet is something that overall should be improved.

If the affinity of the surface would depend on the bump map, then it would probably give the meandering of the water droplet a much more realistic and natural look. The method should be extended so that the droplet adheres to the surface and when the adhesion becomes small enough the droplet will depart from the surface.

Other proposals for extensions can be, to implement the enlargement effect that water droplets have on their underlying surface and how light are reflected in the water droplets. Another extension is implement collision and merging of water droplets.

References

1. Dorsey J, Pedersen H, Hanrahan P. Flow and changes in appearances. *SIGGRAPH 96*, pp. 411-420, 1996.
2. Foster N, Metaxas D. Realistic Animation of Liquids. *Graphical Models and Image Processing*. 58(5) pp. 471-483, 1996.
3. Foster N, Fedkiw, R. Practical Animation of Liquids. *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, 2001.
4. Fournier P, Habibi A, Poulin P. Simulating the flow of liquid droplets. *Proceedings of Graphics Interface 98* pp. 133-42, 1998.
5. Fournier A. A Simple Model of Ocean Waves. *Computer Graphics* 20(4), pp. 75-84, 1986.
6. Kaneda K, Kagawa T, Yamashita H. Animation of Water Droplets on a Glass Plate. *Proceedings of Computer Animation 93*, pp. 177-89, 1993.
7. Kaneda K, Shinya I, Yamashita H. Animation of Water Droplets Moving Down a Surface. *The Journal of Visualization and Computer Animation 10* 1999.
8. Kaneda K, Zuyama Y, Yamashita H, Nishita T. Animation of Water Droplet Flow on Curved Surfaces, *Proceedings of Pacific Graphics 96*, pp. 50-65, 1996.
9. Kilgard M. J. A Practical and Robust Bump-mapping Technique for Today s GPUs *Game Developers Conference, Advanced OpenGL Game Development*. 2000.
10. Max N. Vectorized procedural models for natural terrain: Waves and islands in the sunset. *SIGGRAPH 15*, pp. 317-324, 1981.
11. Nicholson W. K. Linear Algebra With Applications *PWS Publishing Company*, Third Edition, pp. 275, 1995.
12. Peachey D. Modeling Waves and Surf. *Computer Graphics* 20(4), pp. 65-74, 1986.
13. Ts'o P, Barsky B. Modeling and Rending Waves: Wave-Tracing Using Beta-Splines and Reflective and Refractive Texture Mapping. *ACM Transactions on Graphics* 6, pp. 191-214, 1987.
14. Yu Y-J, Jung H-Y, Cho H-G. A new water droplet model using metaball in the gravitational field. *Computer & Graphics 23*, pp. 213-222, 1999.

Distributed Rendering in Heterogenous Display Environments - A Functional Framework Design and Performance Assessment

Seipel S. and Ahrenberg L.,
Department of Information Technology, Uppsala University

ABSTRACT

With this paper we focus on complex display environments in which several users view upon numerous types of 3D displays. We present a method for synchronization and shared state management for independent rendering processes. Our approach is based on TCP/IP based virtual shared memory architecture and intelligent clients in order to accomplish state coherent rendering on multiple displays. We describe a series of benchmark tests that we used to identify frame-to-frame incoherency for rendering of animated objects. The results of these tests allow for a quantitative assessment of the underlying distribution model for networked rendering.

Keywords

Distributed Rendering, NetVR, Display Environments

1. INTRODUCTION

In many applications from industrial design to process control, retrieval of complex information and its appropriate visualization has become a group work task that must be accomplished in a collaborative manner. In the past, different post-desktop computer interfaces have been investigated, which allow for collaborative work in an environment where participants are co-located in the same physical space. Typical examples are the iSpace project at Stanford University [1]. Common to these approaches is that they are based on conventional i.e. 2D human-computer interfaces.

At the Swedish Defense College the potential of collaborative virtual environments has been recognized and research has been initiated (project AQUA) that investigates advanced 3D visualization techniques for command and control [2]. The current configuration of the AQUA visual environment consists of one horizontal large screen display, and four stereoscopic large screen retro-projectors, which are arranged in the corners of the AQUA environment (see figure 1). These displays are viewed by up to 10 users that communicate with one another in the AQUA environment. In addition, each user has at least on local computer display for individual non-collaborative work.

In the AQUA project a general assumption is made that all displays are 3D displays, and that there are arbitrary number and arbitrary spatial orientations of the displays in the physical environment. In consequence all information to be visualized is represented in a thought virtual space that is metrically aligned with the physical space and all displays that are presented in the physical AQUA environment are defined by their corresponding windows-on-world parameterization in the virtual space.

Since the AQUA environment is supposed to be a general 3D visualization environment there is a large number of individual viewing parameter configurations depending

on which user is watching on what display at a certain given point in time.
Rendering of 3D graphics in this multiple-display environment is naturally accomplished by using clusters of independent hosts. Hence, all information that is simultaneously visualized on different screens must be distributed and shared among the involved rendering engines.

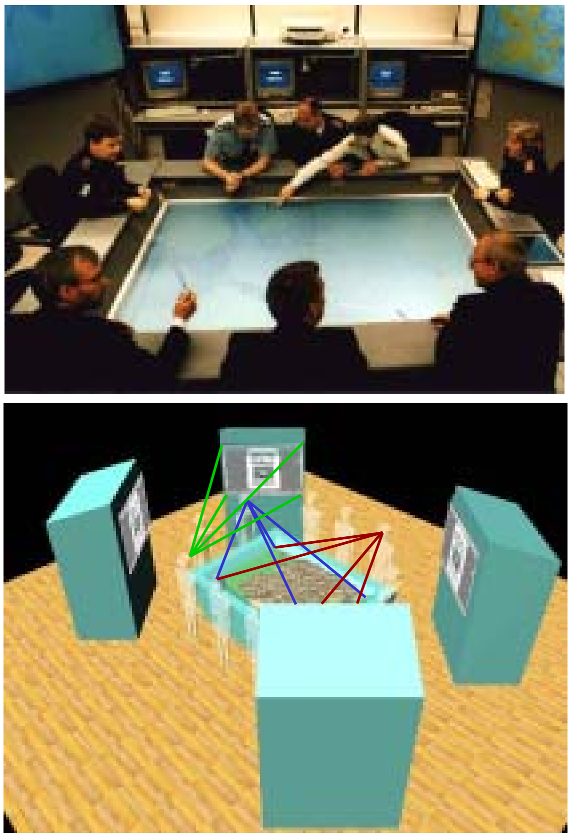


Figure 1. View upon the physical environment of the AQUARIUM (above). For 3D representations various individual viewing frustums must be maintained (below).

2. METHOD

In this project we developed a shared state database for accomplishing simultaneous and distributed 3D rendering of remotely shared virtual objects. It also provides means for runtime re-configuration of the viewing parameters for all involved rendering clients in a complex visualization scenario.
The primary goal of this shared state database is to maintain good runtime performance for a limited number of rendering clients (up to 30) and for a relatively low number of shared states (up to 10.000 floating point values). We also anticipate that rendering applications are executed concurrently in the local area network of a local PC rendering cluster. Unlike other architectures for building distributed VR environments as e.g. DIVE [3], it is not the intention to provide a generalized and fully replicated scene-graph database. We also wanted to avoid a very specialized low level implementation of a specific simulation protocol as e.g. found in the SIMNET environment [4]. Instead, our conceptual approach builds upon intelligent clients that administrate their individual scene-graphs independently. They are using minimal state change propagation to maintain consistency.

On the network level we implemented transparently usable shared memory architecture - STREEP [5]. This library is based on a TCP/IP based protocol and it facilitates allocation of virtual memory, propagation mechanisms for state change updates, and subscriptions for process notification.

Based on this virtual shared memory architecture, applications in the AQUA environment can share relevant information in so-called “pools”. Figure 2 illustrates our concept of shared pools. A pool can be considered as a shared memory area that can be allocated by 3D clients or to which clients can subscribe. A pool contains data of the same type. An example of a pool is a projector pool that contains the parameter configuration to defining the projection pipeline for an individual user looking at a specific display. A pipe pool contains information about the number and configuration of visual channels on a specific display. Other pools are e.g. sensor pools, which store information from various tracking devices, and shared data pool that contains shared data, which is actually to be visualized in the AQUA environment.

Apart from assuring state consistency, the minimization of network latency is one of the most critical issues in distributed rendering. This is in particular true for applications, where the result of the 3D rendering process is visualized simultaneously in the same physical environment. Here, the synchronization of graphical states (i.e. transformation matrices) in the local rendering processes must ideally be frame consistent. In other words, animated objects and other animated states (illumination etc.) must be rendered in the same attitude on all displays at the very moment. Absolute frame consistency cannot be guaranteed with a purely software based synchronization method based on virtual shared memory as in our proposed framework. On the other hand we can ague that absolute frame consistency might not be necessary since the human visual apparatus has limitations both in regard to spatial and temporal resolution.

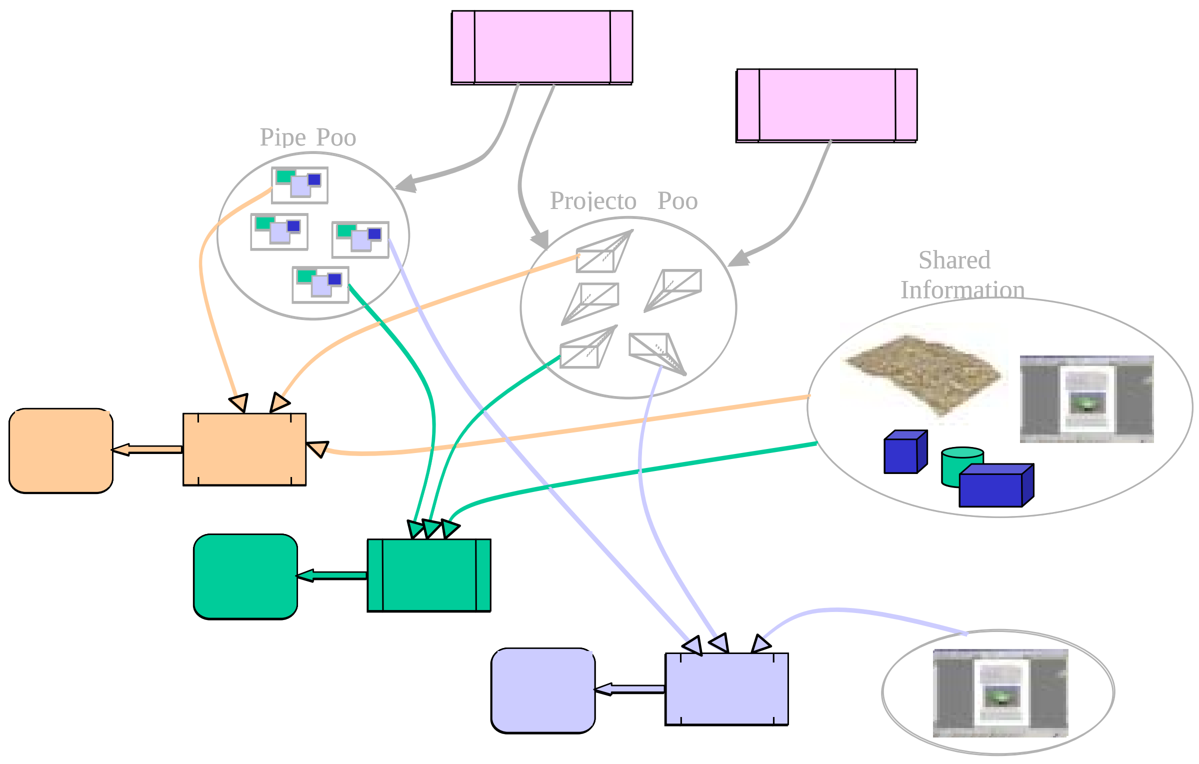


Figure 2. Illustration of the shared data pool concept. Different applications share data for rendering purposes, which is shared a network based shared memory.

In order to shed more light upon this issue we performed an initial runtime performance study. Its goal is to measure and quantify visual artifacts as a consequence of delayed state propagation in our concept for distributed rendering.
To that end, we designed a generic server application that manipulates the states of one (or many) shared objects. Two client applications were designed that render the object with exactly the same viewing conditions. The output of those two independently running client processes is rendered into equally large view ports on the same host computer. Hence, in the ideal case of absolute frame consistency, the graphical output of both processes should be identical for moving objects. In the practical case however, we expect differences in the visual output of these processes as described above. This basic program set-up was used to benchmark the real-time performance under different conditions.

Test set-up:
Two client applications where executed on the same computer, and each was reserved half the available screen space. Both processes where rendering a shared object that was in continuous motion (see figure 3). The test object is a very simple fan geometry rotating with a predefined angular velocity of 360 degrees per second. The actual rotational angle was not manipulated by the client applications, but by a third server process, that also maintains the shared state (rotation) for this fan.

Assessment method and criteria
The client applications where run simultaneously on one computer and a software based frame grabber application (Camtasia by TechSmith) was used to capture the entire screen content i.e. both application windows at a certain frame grabbing rate (20Hz). A sequence of 200 frames was recorded in this manner. In order to assess frame and state delays, we developed a program that superimposes the graphical output of both client applications, and that counts the number off different pixels in those pictures. For absolute frame consistency, the output should be identical and hence the number of differing pixels should be zero. An increasing number of differing pixels indicates frame delay. Since the number of pixels increases in big intervals, every interval indicates the delay of exactly on frame.

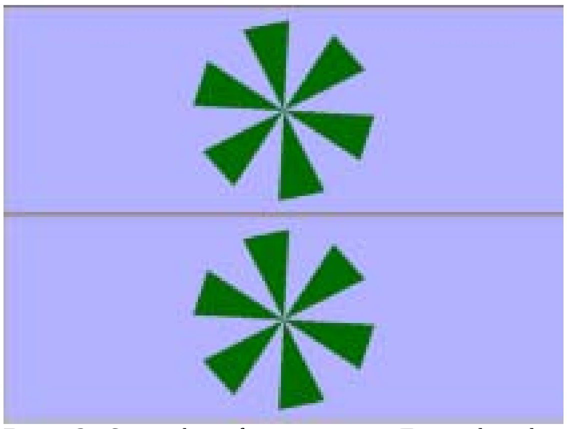


Figure 3. Screen-shot of a test setup. Two independent applications render an animated object, whose rotational state is shared through network based shared memory.

3. RESULTS

Based on the general set-up, we performed three tests under different conditions. Test A: 2 Clients windows are opened on the same computer, the screen resolution is chosen to be 800x600 pixels to allow for higher screen capturing rates. The server is running on a separate remote computer, which resides in the same sub-network. The observed variable is the difference in pixels in the two client windows over time i.e. the number of frames that are out-of-sync. This experiment is repeated several times, whereby the rate of the object updates from the server side is increased. Observe, that an increased update rate on the server side implies relatively small angular increments because the angular velocity was chosen to be constant with 360degrees/sec. The goal is to study the relation between frame delay and shared state update rate.

Table 1 shows the data measured for 200 animation frames. They where measured by capturing at different object update rates. The observed variable is the number of differing pixels in the two client windows, and the data was sorted in descending order. For clarity, only the first 20 data sets are shown in the graphics, because the remaining values are zero altogether. At lower update rates, the angular increment per frame is higher given a constant angular velocity. This explains that in table 1, the difference in the pictures at 10Hz object update rate is higher than compared to the 40 Hz or 50 Hz situations, where there is less angle increment per animation frame.
The figures show that for 10 Hz object update rate there is only 1 out of 200 frames delayed. At 20 Hz and 30 Hz object update rate, there are 7 or 6 frames out of 200 frames delayed by one step. The number naturally increases as the object state update increases, but it does not exceed 11 frames delayed per 200 animated frames.

Interesting to observe is, that even for higher object update rates the number of delayed frames in the client windows does not exceed one frame.

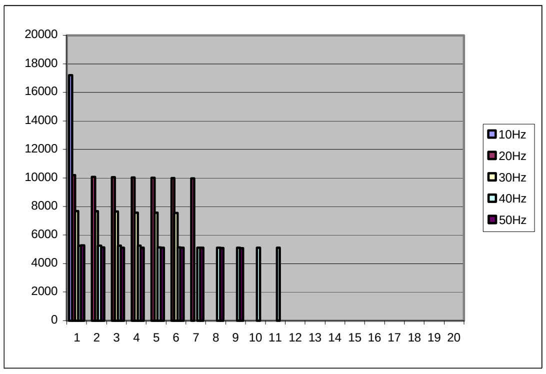


Table 1. The number of differing pixels in between the two client frames sorted by magnitude in descending order for different state update frequencies.

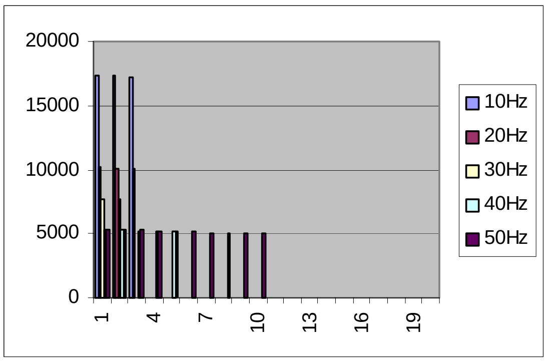


Table 2: The number of differing pixels in between the two client frames sorted by magnitude in descending order for different state update frequencies (server and visualisation clients on same host).

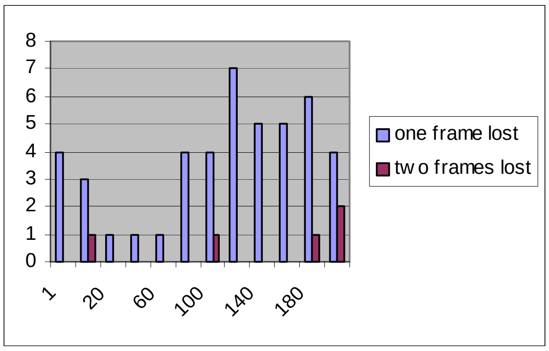


Table 3: The number of in-coherent frames out of 200 frames rendered for increasing number of shared states.

Test B: The test was repeated with the same conditions as in Test A except for the fact that now the server (the process manipulating the object state) is also running on the same host as the two client processes. We would expect that in this condition network routing efforts be reduced, since all TCP/IP traffic is routed on the local host rather than being passed through the Ethernet. Table 2 shows the result for this test. The observed values show a similar pattern as in test A i.e. for increasing state update rates, the number of frames out of sync is increasing. For almost all state update rates, the number of dropped frames was less than in test A. Remarkable is that for the lowest update rate of 10Hz a one-frame delay could be observed three times.

Test C: Another test was carried out to study how much the total network traffic affects frame consistency. In the previous tests, the server only had to maintain the shared attributes of one single object (one fan). In the following experiment the set-up was as in Test A: Two clients windows opened on the same host, the screen resolution set to 800x600 pixels. One is server running on a remote host. The object animation update rate was set to only 10Hz server side. Variable parameters in this test set-up were the number of distributed objects, which was increased stepwise from 1 object to 200 objects. The result of the observations is visualized in table 3.

The result of this experiment shows that up to a number of 100 shared objects there are only between 1 and 4 frames incoherent in an animation sequence of 200 frames. Further increase of the total network traffic (i.e. number of shared objects) will also increase the number of incoherent frames up to seven frame-mismatches at 180 updated objects. Worth mentioning is the fact that with an increasing number of objects that are shared, the frame delay between the two rendering processes is not only one frame, instead the frame incoherence starts to stretch over two animation steps.

4. DISCUSSION

Prior to interpretation of the results the methods of testing must be critically discussed. This initial benchmarking was performed by adopting a very straightforward and easily to implement measuring method using a frame grabber software. This means that the processor load induced by the frame grabbing application introduces artefacts, since less processing power is available to the rendering applications. On the other hand, this speaks rather for an optimistic interpretation of the measured results, because the frame grabber application could indeed be a reason for observed frame delay, which would not occur without running the grabbing process. Due to the need for simultaneously capturing the rendering result of two processes, both rendering clients were running on the same single processor computer system. Load balancing is managed by the operative system and is therefore an uncontrollable factor in the test. Asymmetric task scheduling might therefore be an additional cause for frame inconsistency. After all, the characteristics of the test set-up suggest that we actually should expect better network throughput performance and therefore we do not see a reason to mistrust or pessimistically interpret our results.

5. CONCLUSION AND FUTURE WORK

Summarizing our first observations we can conclude that the TCP/IP based propagation of shared states performs surprisingly well. In a local area network we can expect that about 5 frames in a sequence of 200 animation steps will be out of synchronisation for an average object update rate of 30 Hz (which appears sufficient for almost all object animations in a 3D scene). These frame incoherencies mean only delay of a single animation step. It depends actually on the speed of simulated objects and the scale how dominant this one-frame incoherence is perceived by the user. In regard to the total network traffic we can see that for 200 shared states the proportion of dropped frames is about 10 out of a total of 200 animated frames i.e. 5% of all animated frames are not synchronised. At this stage we can conclude that the runtime performance exceeds our expectations and is more than satisfactory for applications, where a limited number of shared states (below 1000) needs to be synchronized.
It remains to be seen in our future studies, if and how these figures will improve when rendering processes are running exclusively on dedicated hosts. The will also show how performance will scale for increasing numbers of shared states. Our future studies will therefore incorporate superimposition of multiple independent computer displays that will be captured using high-speed video cameras rather than software-based capturing of several application windows that are executed concurrently on one single host. Another issue that needs to be addressed is the question to what extend the human user is capable of perceiving dropped frames or frames that are not fully synchronized. In order to explore that, further user-oriented tests will be carried out to measure visual-perceptual artefacts in different situations of state incoherency.

REFERENCES

[1] Fox, Armando, Brad Johanson, Pat Hanrahan, and Terry Winograd, Integrating Information Appliances into an Interactive Space, *IEEE Computer Graphics and Applications* 20:3 (May/June, 2000), 54-65.

[2] Sundin C. and Friman Henrik (eds.) ROLF 2010 – The Way Ahead and The First Step, Försvarshögskolans Acta C6, Elanders Gotab, Stockholm 2000

[3] Carlsson, C. and Hagsand, O. DIVE – a Multi-User Virtual Reality System. *IEEE VRAIS*, Sept 1993.

[4] Pope, A. The SIMNET network and protocols.*Technical Report 7102*. Cambridge, MA: BBN Systems and Technologies, July 1989.

[5] Lindkvist M: A state sharing toolkit for interactive applications. *Master Thesis.* Department of Information Technology, Uppsala University. 2001.

Real-Time Image Based Lighting in Software Using HDR Panoramas

Jonas Unger, Magnus Wrenninge, Filip Wänström and Mark Ollila
Norrköping Visualization and Interaction Studio
Linköping University, Sweden

Abstract

We present a system allowing real-time image based lighting based on HDR panoramic images. The system performs time-consuming diffuse light calculations in a pre-processing step, which is key to attaining interactivity. The real-time subsystem processes an image based lighting model in software, which would be simple to implement in hardware. Rendering is handled by OpenGL, but could be substituted for another graphics API. Applications for the technique presented are discussed, and includes methods for realistic outdoor lighting. The system architecture is outlined, describing the algorithms used.

Introduction

Over recent years there has been much excitement about image based rendering, modelling and lighting [1,3,4,5,6,7]. The aim of this research is to create a system that allows dynamic objects to be lit accurately according to their environments, preferably using a global technique (i.e. everything in the surrounding scene can affect the given object), while still keeping the process fully real-time. The main benefit will is to obtain greatly enhanced realism, as well as better visual integration through the added object-scene interaction. This aim requires us to part with the ordinary way of doing real-time lighting; that is, with well-defined light sources. In real life, everything we are able to observe reflects light to a certain extent, and thus cannot be neglected in lighting calculations [1, 5]

Disregarding the fact that other objects cast light onto each other is a great simplification, and the greatest limitation of real-time rendering has always been the necessary use of local illumination models. Local illumination accurately models what a given surface would look like considering one or a few point light sources, but does not take into account the interaction between illuminated surfaces. The downside of this is evident in the poor realism seen in most computer games and other real-time applications.

In the last few years the use of lightmaps has become popular, since they allow static objects to have a radiosity solution baked onto the model as a texture. This way, the time-consuming part of the lighting calculations is moved to a pre-processing step. Lightmaps work very well for buildings and such objects, but do not allow dynamic objects to be lit using global illumination. The fact that dynamic objects (for example player characters and opponents) are not lit as convincingly as their environments makes for poor and unrealistic integration.

Applications

Our technique of using images to light virtual scenes has many possible application areas. Here we will focus on lighting outdoor scenery. Lighting outdoor scenes has always been a difficult area in creating computer images and especially in real-time graphics. As mentioned earlier, this is a consequence of using local illumination models. Mainly, these fail to model the fact that in outdoor scenes a large contribution of incoming light is reflected light, whereas local illumination models only regard direct lighting, i.e. light sources that have a direct view of the object.

By capturing the light situation using a light probe [5], far more realistic lighting can be achieved. This kind of lighting has proven successful in, for example, motion pictures. All kinds of natural phenomena in outdoor scenes could potentially benefit from using image based methods for lighting.

The System

Figure 1 shows the basic dataflow of the system. As can be noted, it is divided into two main blocks – offline image pre-processing and online real-time rendering.

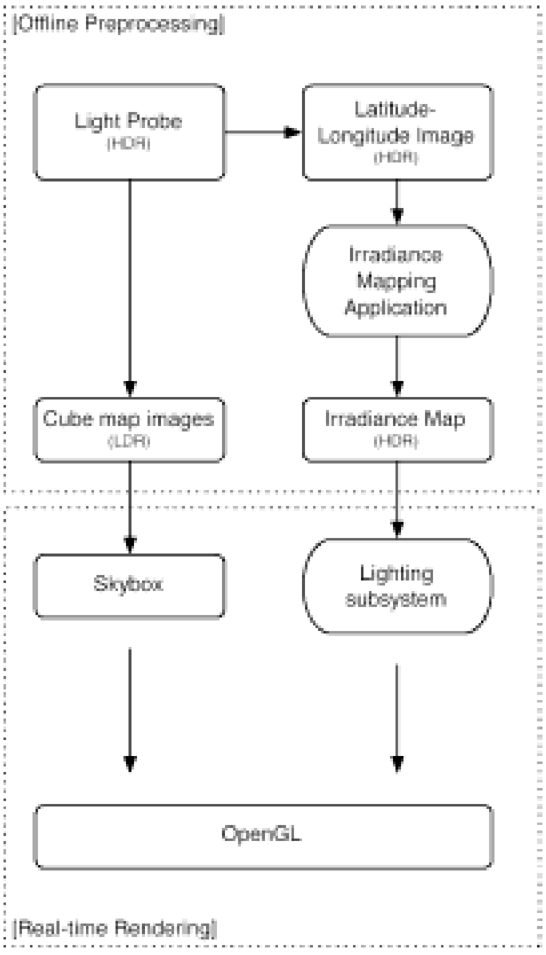


Figure 1: System architecture

Irradiance Mapping

In order to achieve real-time lighting, the time consuming integrations are performed in a pre-processing step. The HDR panorama image (the radiance map) is used to calculate a diffuse map which, during rendering, is used as a lookup table by mapping a normal direction **N**(λ,χ) into an irradiance value I_N(**N**). Both maps are stored as a latitude-longitude map, since that gives a one-to-one correspondence with spherical coordinates.

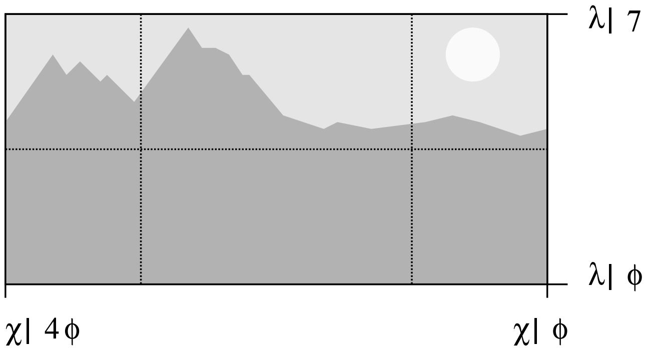


Figure 2: Radiance map.

The radiance map (see Figure 2) is a representation of the plenoptic function I(*x*, *y*, *z*, λ, χ, ς, *t*) as $I_R(\lambda,\chi)$. We assume that the interaction between an object at point **P** = (*x*, *y*, *z*) and the distant scene recorded in the irradiance map is one-way, i.e. we only consider light transport *from* the surrounding scene to the object and not vice versa. This is accurate as long as the distance between the object and the surrounding scene is sufficiently great.

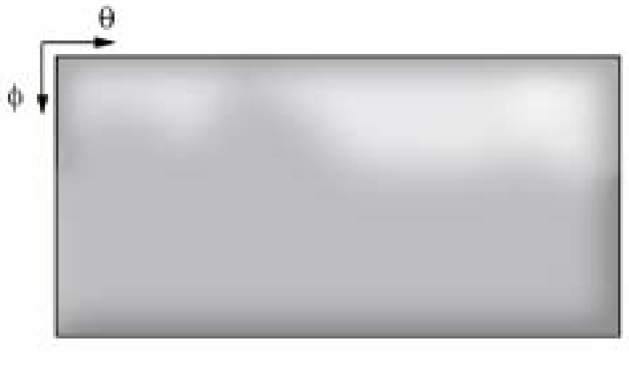


Figure 3: Diffuse map.

The diffuse map (see Figure 3) can be explained as follows: For every direction (λ,χ), given the point **P** where the radiance map was captured, there is an imaginary surface element with a normal **N**(λ,χ). The diffuse map then stores a measure of the incoming radiance onto the surface element for each direction **N**, which is calculated in as follows:

$$I_{\mathbf{N}} | \int_{4\phi}^{\phi}\int_{0}^{\phi/2} I_R(\mathbf{L})\,(\mathbf{N} \notin \mathbf{L})\,d\lambda d\chi$$

$$\mathbf{L} \,|\, (\lambda,\chi),\ |\mathbf{L}| \,|\, 1$$

Where **L**# is the unit vector in direction **(**λ,χ**0**# where# I$_R$**(** $\breve{L}$ 0 is the intensity in the radiance map in direction **L** and **N** is the surface element normal. In Figure 4 we show how the integration limits comes from the hemisphere around the normal **N**. To produce an entire diffuse map, solve I$_N$ for all **N** and store the result at the pixel corresponding to **N**5

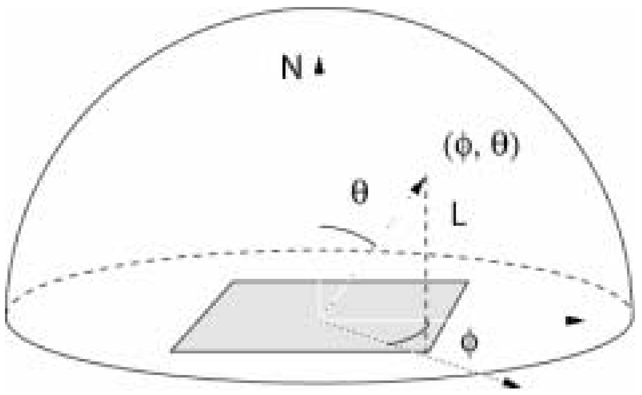


Figure 4: Hemisphere

Lighting

During rendering, the normal direction **N** of any given vertex in the object is used to look up the diffuse light intensity affecting it. Since the intensity values in the diffuse map are stored in spherical coordinates this is a straightforward process. The value is then used in combination with the vertex material properties to produce the final colour at the vertex. The algorithm is outlined below:

```
    For each vertex
        Get the normal N
        Convert N to spherical coordinates N = (λ,χ0#
#   #   Look up the pixel value stored at (λ,χ0 in the diffuse map
        Light vertex according to material properties and the light intensity
        Store result as vertex colour
    End
```

Results

The system presented fulfils the goals set, being real-time and interactive. It implements a software image based lighting model, passing lit geometry to OpenGL for rendering. Inputs required are 3D-objects and HDR panoramas, making the process simple and straightforward. The panoramas are processed off-line in our custom image-processing application to produce diffuse maps. This pre-processing removes the time-consuming integration involved in global illumination solutions allowing real-time performance during rendering.

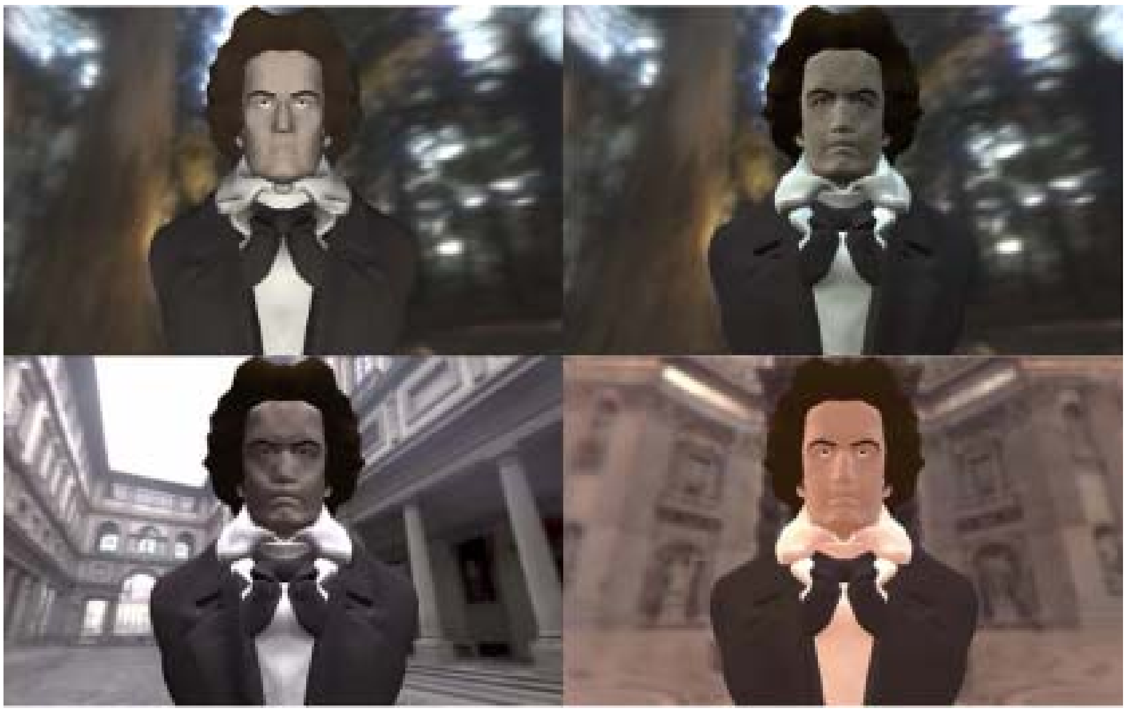


Figure 5: We demonstrate the difference between using a single point light source and using the algorithm described in this paper. Most importantly, the second image shows realism in two areas: matching light intensity with that perceived as coming from the surroundings, as well as matching the overall colour balance of the object to the panorama. We show two images which show the effect of colour blending. Great dynamism is evident in the second image where the light under the arcades is approximately 1000 times weaker than that coming from the somewhat cloudy sky. This gives rise to the truly dramatic, yet realistic, shading of the model.

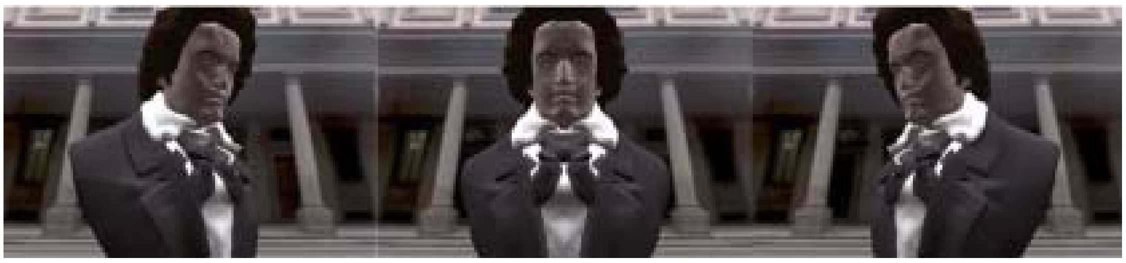


Figure 6: The dynamic properties of the system are shown in the above 3 images. Here, the model is clearly lit differently as it is spun to face different parts of the surrounding scene. The effect is clearly evident in the forehead of the model, as well as on its coat.

Conclusions and Future Work

Further developments of this lighting technique can be divided into three main areas: improving the quality and speed of the rendering through hardware implementation, adding greater realism to the given system, and extending the system to allow completely dynamic scenes and views to be rendered [2].

In order to speed up the rendering process even further it could be implemented for consumer hardware with vertex and pixel programs. This would also allow the lighting to be calculated per pixel instead of per vertex, as is currently the case. The entire lighting process could then be performed using just a single texture unit, however this requires implementation of HDR texture support in hardware, as well as reprogramming the hardware's texture lookups.

Yet greater realism could be achieved by adding reflections and specular highlights. Both could be pre-processed by the same image-processing software as used to create diffuse maps. This way, rendering quality close to what ray-tracing software produces is possible for dynamic objects, not just static ones. The system described is currently only able to display lighting situations in a single location. This would be quite sufficient to model outdoor lighting coming from the sun and clouds. A major advancement would be to extend the system into using many light situations. This would allow dynamic actors in a scene to be influenced by different light situations depending on where in a scene they are located.

Acknowledgements

Paul Debevec for providing the light probe images from http://www.debevec.org/.

References

[1] CHEN, W.-C., BOUGUET, J.-Y., CHU, M. H., AND GRZESZCZUK, R. 2002. Light field mapping: efficient representation and hardware rendering of surface light fields. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, ACM Press, pp 447–456.
[2] DAMEZ, C., DMITRIEV, K., AND MYSZKOWSKI, K. 2002. Global illumination for interactive applications and high-quality animations. In *Eurographics 2002 STAR Reports*, 1–24.
[3] COHEN, J., TCHOU, C., HAWKINS, T., DEBEVEC, P. 2001. Real-Time High Dynamic Range Texture Mapping. *Rendering Techniques 2001: 12th Eurographics Workshop on Rendering*. pp. 313-320, 2001.
[4] DEBEVEC, P. E. Rendering synthetic objects into real scenes. 1998. In *Proceedings of the 25th annual conference on Computer graphics.* ACM Press..
[5] DEBEVEC, P. E., AND MALIK, J. Recovering high dynamic range radiance maps from photographs. 1997. In *Proceedings of the 24th annual conference on Computer graphics*, ACM Press, pp. 369-378.,
[6] DEBEVEC, P. 1998. Rendering synthetic objects into real scenes: bridging traditional and image-based graphics with global illumination and high dynamic range photography. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, ACM Press, 189–198.

Towards a Perceptual Method of Blending for Image-Based Models

Gordon Watson, Patrick O'Brien† and Mark Wright

Edinburgh Virtual Environment Centre
University of Edinburgh
JCMB, Mayfield Road,
Edinburgh EH9 3JZ
<g.c.watson, mark.wright>@ed.ac.uk

†Now at Sony Computer Entertainment Europe
patrick@optikal.demon.co.uk

Abstract
We present a technique for view-dependent texture mapping that attempts to minimise the perceived artefacts that can arise from image blending. These artefacts result from imperfect image registration, typically due to un-representative scene geometry, or scene motion during capture. Our method draws inspiration from work in image mosaicing, but uses a metric based on perception of Mach bands that we also use to quantitatively evaluate the method. We discuss the implications of this work to creating a fully perception-based method of image-based rendering.

Keywords: image blending, image-based rendering, perceptual methods.

1 Introduction

The image-based rendering field of computer graphics attempts to produce novel views of a scene by reconstructing the plenoptic function from discrete samples [1, 10]. The samples usually take the form of photographs of the scene, and reconstruction involves sample interpolation. The approach has several advantages over traditional three-dimensional graphics, including constant rendering cost regardless of scene complexity and photorealistic output.

A key operation in IBR is therefore interpolating the discrete samples to produce a new approximation of the plenoptic function for the novel view. To date, image-based rendering methods have employed interpolation schemes with simple mathematical formulations that pay little attention to how the resulting images are actually perceived. We believe an interpolation scheme based upon image perception would both improve the quality of rendered images and extend the scope of image-based rendering, for example, to the capture and rendering of natural outdoors environments.

In this work we focus on a specific image-based rendering technique referred to in [4] as view-dependent texture mapping (VDTM). A geometric model of the scene (termed a geometric proxy by Buehler et al. [2]) is recovered from the input images using photogrammetric modelling techniques. The geometric proxy is texture mapped by projecting the photographs back onto it, producing novel views of the scene, VDTM assumes only an approximate geometric model of the scene is available but can give the appearance of more complex geometry being present.

In common with other image-based rendering techniques, VDTM employs either linear interpolation between the images, or nearest-neighbour.

We will show that in VDTM these interpolation techniques can generate artefacts when geometric features in the real scene are not modelled by the proxy. A highly accurate proxy is difficult to recover from a sparse set of input images, so we seek to reduce these artefacts by controlling interpolation using a perception based metric.

The metric allows us to classify sample components and interpolate them at different rates thus reducing interpolation artefacts. We use the same perception based metric to verify the quality of our results.

2. Background and Related Work

IBR attempts to model a scene using a continuous representation of the plenoptic function [10]. The plenoptic function was first proposed by Adelson and Bergen in [1] and describes all the rays of any wavelength that are visible from a point in space.

Producing a continuous representation of the plenoptic function requires some way of processing discrete samples of the function to provide a reasonable approximation. Many methods of doing this have been proposed. For example, Lippman's Movie Map technique [8] simply chooses the sample which is closest to the current viewing position (equivalent to nearest neighbour interpolation). Other methods are based on interpolating the samples using image flow fields [3, 7, 10].

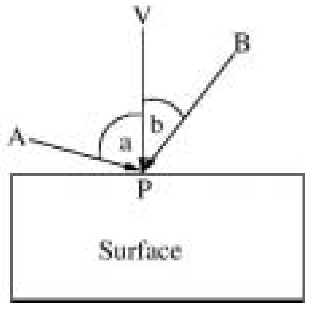


Figure 1: The correct choice of photograph to use for texturing point P depends on the relative positions of the real cameras A & B, and the virtual camera V.

View-dependent texture mapping, presented by Debevec et al. in [4], uses a fitness function to assess the suitability of each input image for texturing the geometric proxy.

The fitness function considers the relative gaze angles of the viewer and the camera that captured the image under consideration. The smaller the divergence between the angles, the more suitable the image for texturing. Figure 1 illustrates this with two cameras. Intuitively, it is desirable to use the image captured by camera B. The fitness function weights each image's contribution to the final colour of the point being textured.

In [2], Buehler et al. present an IBR algorithm which generalizes many existing IBR approaches. With a sparse set of input images and an accurate geometric proxy, the algorithm behaves like VDTM. This approach uses a more advanced fitness function, considering sample resolution in addition to angle similarity.

Both [4] and [2] use the fitness function to linearly interpolate the samples. We will show that when the geometric proxy is approximate (as is often the case due to the low number of input images), piecewise linear interpolation generates artefacts.

Pollard et al. have shown that IBR artefacts can be reduced by interpolating different texture frequencies at different rates [11]. In this paper we examine the nature of these artefacts and extend the technique. We use a metric based on the human visual system to identify the texture components which should be blended at different rates. The new metric is then used to evaluate the quality of results.

2.1 The Mach Band Effect

An edge-ramp in an image corresponds to high spatial frequencies. The human vision system is well adapted to identifying these high frequencies that cause a perceptual phenomenon known as the Mach band effect (discovered in 1865 by Mach, an Austrian physicist), which emphasizes the border between two regions of differing intensities. Close to the border, the region on the light side looks lighter and the region on the dark side looks darker. These regions only appear close to the border, and they are called Mach bands. A detailed study of Mach bands can be found in [12].

Mach bands are relevant in computer graphics because the eye is naturally drawn to the regions containing them. Of course, Mach bands in a computer generated image are not necessarily artefacts because we see them around edges in the real world. However, a Mach band may greatly emphasize existing artefacts. For example, in computer graphics, smoothly curving surfaces are often approximated using polygons. When the polygonal surface is illuminated the polygons in the surface will have slightly different intensities due to their differing orientations relative to the light source. These differing intensities produce obvious edges between the polygons producing a faceted appearance. The edges are emphasized by Mach bands and the illusion that we are viewing a smoothly curving surface breaks down. The obvious solution of increasing the number of polygons fails since it increases the number of high frequencies in the image, and therefore creates more Mach bands.

Gouraud showed that interpolating the vertex shading values allowed smooth shading across polygon borders. This eliminates intensity discontinuities and minimises Mach bands [5].

3. Existing Interpolation Techniques

3.1. Why Interpolate ?

Images of a scene are discrete samples of the plenoptic function. As described in section 2, the aim of image based rendering is to reconstruct a continuous representation of the plenoptic function from samples. In our case, the reconstruction takes place in the texture blending stage of view dependent texture mapping; the most appropriate samples are selected and interpolated to give an approximation of the plenoptic function for the current viewpoint.

3.2. Nearest Neighbour Interpolation

The easiest way to achieve the reconstruction is to use '*nearest neighbour*' interpolation. This simply means using the sample with the highest *fitness* for the current point. That is, choose the sample whose parameters (such as view direction and position) most closely match the novel view parameters.

Clearly, this means that the samples will not be blended. When the viewer moves, the most suitable sample may change, causing the surface texture map to be instantly swapped for the new 'nearest neighbour'. When this form of interpolation is used, the texture changes are very obvious due to incongruent textures. The causes of texture incongruence are explored in section 4.

3.3. Piecewise Linear Interpolation

In approaches such as [4] and [2], the plenoptic function is reconstructed using piecewise linear interpolation of the samples. Multiple textures may be blended together to give the colour for a surface point, using the fitness of an image to weight its contribution. For example, imagine we are texturing point P in figure 1 using images captured from viewpoints A and B. Call these Sample α and Sample β respectively. As the viewer moves from viewpoint A towards viewpoint B, Sample α will contribute progressively more to the point, while Sample β will contribute progressively less. When the viewer is halfway between A and B, each sample will contribute equally to the colour of the point.

Piecewise linear interpolation produces smooth changes between textures as the viewpoint changes and avoids the sudden texture switching and 'seams' which occur with nearest neighbour interpolation. However, texture incongruence may still cause artefacts with this interpolation technique.

4. Sources of Texture Incongruence

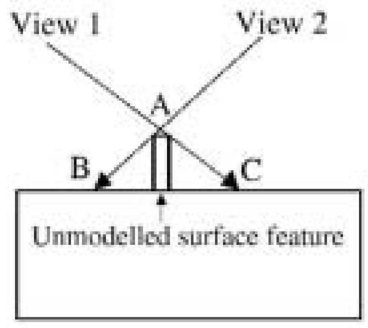


Figure 2: Depending on the viewing position, point *A* projects to a different point on the surface.

4.1 Unmodelled geometric features.

Figure 2 illustrates what happens when a photograph is taken of a surface with a protrusion. Depending on where the photograph is taken from, the protrusion will project onto a different position on the surface. If a perfect geometric proxy has been recovered, the photographs will project back onto the model in the correct manner when texture mapping is performed.

Unfortunately in practice it is not possible to recover a perfect geometric model of the scene. If the protrusion in figure 2 is not recovered, the part of the texture containing the protrusion will be projected onto the model surface behind it. The exact position on the surface where it appears depends on the viewpoint of the original photograph. Therefore photographs taken from different positions will disagree as to where the protrusion appears.

4.2 Non-lambertian Surfaces.

In the real-world many surfaces exhibit non-lambertian reflections. If a surface is slightly specular, it may appear different shades when viewed from different angles. As a result, two different photographs of the same surface under the same lighting conditions may record dramatically different colours for the surface.

4.3 Motion during capture.

In practice it is not possible to capture all the images required for an image-based model instantaneously. This implies motion may occur during capture, either to elements of the scene, or to the position or brightness of the light sources. This is a very real problem in the case of capture out of doors.

Changes in the brightness of a light source will cause artefacts similar to a non-lambertian surface, wheras motion of the scene itself will cause an artefact similar to un-modelled scene geometry – due to the images disagreeing as to the location of a scene element.

5. Implications for Blending

Nearest Neighbour Interpolation always produces the same class of artefact regardless of the cause of texture incongruence. The artefacts take the form of 'seams' on the model surface at the border between the textures. In a movie sequence, this causes an effect known as 'popping', when the current texture is abruptly changed for the new 'most fit' texture.

Piecewise Linear Interpolation copes well with texture incongruence caused by slightly specular surfaces. The transition from one texture to another is smooth with no obvious seams or popping. Texture incongruence caused by unmodelled geometry is not handled as effectively. If the two photographs in figure 2 are linearly interpolated as the viewer moves from View1 to View2, an image of the unmodelled protrusion will be simultaneously projected onto two different places on the surface. The edges in the textures being interpolated will not align, even though they are associated with the same edge in the real scene. This creates spurious high frequencies in the synthetic image, which generates undesired Mach bands. The artefacts will be at their worst when the viewer is halfway between view1 and view2, an example of this can be seen in figure 3b.

6. The Mach Band Metric.

As previously described, linear interpolation of textures that are incongruent due to unmodelled geometry produces spurious high spatial frequencies, and hence more Mach bands. The presence of spurious Mach bands can dramatically alter the perceived quality of a synthetic image [5].

If n textures are to be blended, each containing m Mach bands, the ideal situation is one in which no texture incongruence occurs – i.e the Mach bands from each image align perfectly with each other. This produces a synthetic image with m Mach bands Conversely in the worst case scenario an image with $n*m$ Mach bands could be generated in the case that no Mach bands align at all between the textures.

We propose that measuring the number of generated Mach bands in a synthetic image provides a perception based metric for comparing the relative proficiencies of different interpolation techniques. Clearly, as the Mach bands tend to m, less artefacts will be generated. An obvious way of minimising these is to use nearest neighbour interpolation. However, as previously discussed, this solution can suffer from significant artefacts in the form of `seams' on the model surface.

6.1. Desired Properties of a texture interpolation technique.

This motivates the following requirements for a texture interpolation technique:

1. Produces smooth changes between textures for non-Lambertian surfaces
2. Minimises the number of Mach bands around areas of unmodelled geometry.

7. Frequency Dependent Interpolation

We attempt to meet the requirements stated in section 6.1 by separating a texture **t** into two parts:

1. The high frequency (ie Mach band generating) component (call this $\mathbf{t_m}$).
2. The component containing no high frequencies (call this $\mathbf{t_n}$)

These components are then interpolated at different rates.

If the $\mathbf{t_m}$ components are interpolated using a nearest neighbour scheme, the high spatial frequencies from incongruent textures cannot be simaultaneously projected onto the same point, thereby minimising spurious Mach bands. Linear interpolation of the tn components produce smooth overall changes between textures.

We call this technique Frequency Dependent Interpolation. It requires a method of identifying the regions that cause Mach bands, a method for splitting the texture into the two regions, and a method of recombing the texture components on the model surface.

7.1 Identifying Mach band regions

Marr-Hildreth edge detection [9] was used to identify the high spatial frequencies that give rise to Mach bands. The Marr-Hildreth method is isotropic and closely matches the human perception of edges. The filter is equivalent to applying a high-pass filter to the texture, with the cut-off set to the response of the eye.

The result was thresholded to provide a binary map of high frequency edges.

7.2 Separating texture components.

We require a process that separate the texture into tm and tn components such that $\mathbf{t_m} + \mathbf{t_n} = \mathbf{t}$.

Marr-Hildreth filtering identifies those pixels that give rise to Mach bands. The image is dilated to form a small region around each edge, gaussian filtering is performed inside that region to remove all high frequency components. The size of the dilation filter depends only on the texture size and is simply required to give adequate support to the gaussian filter.

This result is now $\mathbf{t_n}$ and contains no Mach bands, $\mathbf{t_n}$ is then subtracted from $\mathbf{t}$ to form $\mathbf{t_m}$, which is re-scaled into a signed 8-bit image.

7.3 Texture recombination.

Now that we have $\mathbf{t_n}$ and $\mathbf{t_m}$ for each texture we must interpolate all the components in such a manner that minimises Mach bands and ensures a continuous surface colour. We use a fitness function related to the angular difference between the sample camera and the viewing position. The low frequency components are linearly interpolated using these fitness weights, wheras the high frequency components are interpolated using a nearest neighbour scheme that ensures only a single set of Mach band producing edges appear on the final image at any time.

8. Implementation.

A renderman complient ray-tracer called the *Blue Moon Rendering Tools* was used to generate the synthetic images. Textures were projected into the scene using custom light source shaders which could be configured with the intrinsic and extrinsic parameters of the camera that captured the original image. Two texture projectors replaced each camera from the original scene, one projecting tm and the other projecting tn. In addition a third projector was used to project an *alpha* channel. This allowed the weight of each image to be reduced towards the edge and allowed occluding foreground objects to be masked out.

Interpolation was achieved using custom surface shaders assigned to the model surface. Surface shaders which performed nearest neighbour, linear, and frequency dependent interpolation were implemented. This allowed different algorithms to be easily compared.

9. Results

Figure 4 shows an original image and the corresponding low and high frequency components after filtering. Figure 3a shows a frame from an animation rendered with frequency dependent blending, linear blending is shown in figure 3b. Figure 5 shows a plot of Marr-Hildreth filter response – a proxy to Mach band number, for a simple cross-fade between two images. As can be seen the frequency-dependent method reduces the overall number of Mach bands compared with the linear method except in the transition region where the two are equal.

10. Discussion & Conclusions.

The results have showed a blending method based on perception of edges can reduce visible artefacts for an image based model.

However seams still appear in the images. These are due to the algorithm 'cross-fading' between the two highest weighted images, the seam representing a transition from one '2nd best' image to another. The problem occurs if the capture cameras are not arranged in a plane. Cross-fading between more than two images reduces the appearance of visual seams, but at the expense of more cumulative errors and more Mach bands.

The solution to this problem isn't clear with the current implementation since the ray-tracer employed casts rays that are completely independent from each other, and hence a point on the surface has no knowledge of adjacent surface points. This may be a computationally desirable property for parallel execution, but it means we are not able to take into account *visual field* perception or 'hole fill' areas unseen by any camera.

Animation frames generated by the system are similarly independent from previous or future frames meaning we are unable to take into account *temporal* perception and coherence.

Clearly the eye does not perceive the world in this manner, our perception-based approach to rendering therefore raises the question as to what would be the requirements for a renderer to use a comprehensive model of visual perception, and the challenge of implemention - particularly in an interactive context.

11 Acknowledgements

The authors wish to thank Dr Paul Debevec for access to the Facade photogrammetric modelling system.

12 References

[1] E. Adelson and J. Bergen. *Computational models of visual processing*, chapter 1. The MIT Press, Cambridge Mass. 1991.

[2] C. Buehler, M. Bosse, S. Gortler, M. Cohen, and L. McMilllan. Unstructured Lumigraph rendering. *Proceedings of SIGGRAPH 2001.* 2001

[3] S. Chen and L. Williams. View interpolation for image synthesis. *Proceedings of SIGGRAPH 1993,* 1993.

[4] P. Debevec, C. Taylor, and J. Malik. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. *Proceedings of SIGGRAPH 1996*, pages 11-20, 1996.

[5] H. Gouraud. Continuous shading of curved surfaces. *IEEE Transactions on Computers*, pages 623-629, 1971.

[6] L. Gritz and J. Hahn. Bmrt: A global illumination implementation of the renderman standard. *Journal of Graphics Tools*, 1(3), 1996.

[7] S. Laveau and O. Faugeras. 3d scene representation as a collection of images and fundamental matrices. *INRIA, Technical Report* No. 2205, 1994.

[8] A. Lippman. Movie-maps: An application of the optical video-disc to computer graphics. *Proceedings of SIGGRAPH 1980*, 1980.

[9] D. Marr and E. Hildreth. Theory of edge detection. *Proceedings Royal Society of London Bulletin*, 207:187-217, 1980.

[10] L. McMillan and G. Bishop. Plenoptic modelling: An image-based rendering system. *Proceedings of SIGGRAPH 1995*, 1995.

[11] S. Pollard and S. Hayes. 3d video sprites. *Report HPL-98-25*, 1998.

[12] F. Ratliff. Mach Bands: Quantitative Studies on Neural Networks in the Retina. *San Fransisco: Holden-Day*, 1965.

[13] Zakia, Richard D. Perception and Imaging. *Focal Press,* 2002.

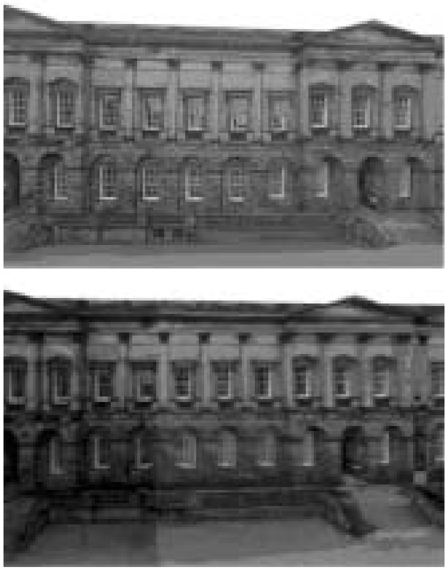


Figure 3: A frame blended using frequency-dependent blending at the top, and linear blending on the bottom. Looking at the perspective at the two ends of the building it is evident they are derived from different source photographs. The geometric proxy for this façade is just a plane.

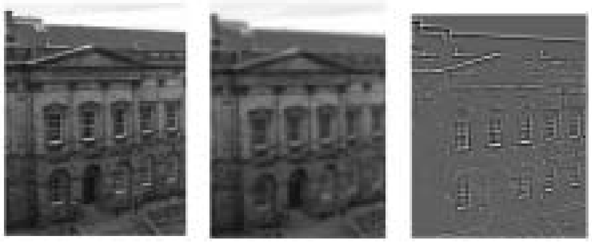


Figure 4: The original image is shown on the left, the low-frequency image in the middle, and the high frequency image (normalised) on the right.

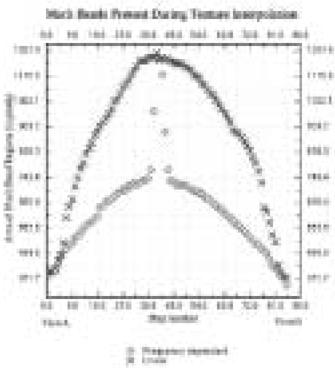


Figure 5: A plot of Marr-Hildreth filter response shows the frequency-dependent interpolation reduces the appearance of Mach bands except for the centre 'change-over' region

Work in Progress

Framework for Real-Time Simulations

Anders Backman
VRlab/HPC2N

Abstract

Virtual Environments (VE) as well as games has traditionally been pre-scripted and or pre-animated. Interacting with an animated environment can be hard due to the explicit actions that have been introduced into the environment.

With more and more CPU power and recent breakthroughs in real-time physical simulation methods, it now has been possible to incorporate complex physical structures in the VEs.

This talk will describe a work in progress at VRlab, where the aim is to develop a simulation framework for interactive real-time physical simulations. It will also touch the benefits of using real-time physical simulation when it comes to authoring of the VE as well as interacting in it.

An application, the Haptic Wheelchair will be presented, using the existing framework.

SIGRAD (2002)
Mark Ollila (Editors)

Surface Construction with Near Least Square Acceleration based on Vertex Normals on Triangular Meshes

Tony Barrera

Cycore AB
Dragarbrunnsgatan 35, P.O. Box 1401, S-751 44 Uppsala, Sweden

Anders Hast

Creative Media Lab
University of Gävle, Kungsbäcksvägen 47, S-801 76 Gävle, Sweden. aht@hig.se

Ewert Bengtsson

Centre for Image Analysis
Uppsala University, Lägerhyddsvägen 17. S-752 37, Uppsala, Sweden. ewert@cb.uu.se

Abstract
Shading makes faceted objects appear smooth. However, the contour will still appear non smooth. Subdivision schemes can handle this problem by introducing new polygons in the mesh. The disadvantage is that a more complex mesh takes more time to render than a simple one. We propose a new method for constructing a curvilinear mesh using quadratic curves with near least square acceleration. This mesh could be used for subsequent subdivision of the surface. This can be done on the fly, at least in software rendering, depending on the curvature of the contour. The advantage is that new polygons are only inserted where needed. However, in this paper we will focus on how such curvilinear mesh can be constructed using vertex points and vertex normals for each polygon. Thus, information about neighboring polygons are not needed and on the fly subdivision is made easier.

1. Introduction

Recursive surface subdivision has gained a lot of attention in recent years since it makes it possible to have several levels of detail[11]. As an example, the famous Utah teapot was modeled by using Beziér patches in a rather coarse mesh. This mesh could be subdivided into smaller patches creating a tighter mesh, and thus a smoother looking teapot. The new mesh points are created by computing new points on each Beziér surface patch. Nevertheless, many models do not have an underlying curve description that could be used for subdivision. Such faceted models only have a flat polygon mesh. The interior of the object can appear smooth if a shading algorithm is used. Nonetheless, the contour will still appear non smooth since the edges are straight lines. However, it is possible to create a spline surface interpolating or approximating the polygonal mesh, which then can be used for recursive subdivision. This will yield a tighter mesh, which will make the object appear more smooth on the contour.

Several such approaches have been made and the algorithm by Catmull and Clark[2] use information about neighboring polygons. Others only use the vertices and normals at these vertices, when constructing the new surface. Volino and Magnenat-Thalmann[10] construct a spherical surface called a Spherigon. Vlachos et al.[9] use a three sided cubic Beziér surface for their Curved PN triangles and Bruijn[1] uses quadratic Beziér surfaces. Overveld and Wyvill[7] also use Beziér surfaces, however they replace the cubic curve with two quadratic curves in order to guarantee monotonic curvature. In surface subdivision presented by Maillot and Stam[5], a spline surface is generated and the curvature and smoothness is controlled by a parameter α, which is set to some proper value for optimal smoothness. Other subdivision schemes have been introduced by Loop[4] and kobbelt[3].

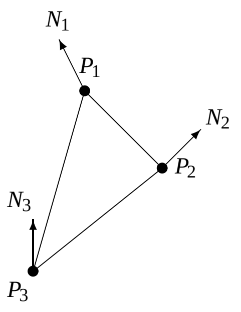


Figure 1: *Three sided polygon with three vertices and three normals*

1.1. Main Contribution

We propose a new algorithm for finding a near least square acceleration curvilinear mesh. And we will focus on three sided polygons for simplicity. This mesh is constructed by using quadratic curves which are derived using vertex normals and vertex points only. Thus, we do not propose a new subdivision scheme. We only propose how the new vertex points can be found by using the curvilinear mesh. Moreover, it will not produce surfaces that are C^1 continuous over polygon edges. This would have required the use of cubic surfaces. However, this method will be faster since it uses quadratic meshes. Hence, the method will yield a surface which is smoother than the original mesh.

2. Second Degree Best-Fit Least Square Acceleration Boundary Curves

The method of finding a least square acceleration boundary curve has been explored before by Overveld and Wyvill[7]. They use cubic Beziér curves that are divided into two quadratic curves. We will show how this can be done for quadratic curves directly, even though the curve will be an approximation of the a least square acceleration curve.

We assume that we have a triangular mesh with coordinate points P_1, P_2, P_3 and corresponding normalized normals $\mathbf{N}_1, \mathbf{N}_2, \mathbf{N}_3$ as shown in figure 1. The tangents corresponding to a pair of normals as shown in figure 2 can be obtained by using the so-called Gram-Schmidt orthogonalization algorithm[6]:

$$\mathbf{T}_1 = \mathbf{N}_2 - \mathbf{N}_1(\mathbf{N}_1 \cdot \mathbf{N}_2), \qquad (1)$$

$$\mathbf{T}_2 = -\mathbf{N}_1 + \mathbf{N}_2(\mathbf{N}_1 \cdot \mathbf{N}_2). \qquad (2)$$

Note. that the normals are assumed to be normalized. We shall show later that we do not need to normalize the tangent vectors. However, at this point we shall assume that they have unit length. It should also be pointed out that when the angle θ between the normals, as shown in figure 2, is zero, then we can not compute the tangent in this way. The simple solution is to use linear interpoltaion between the surface points instead of a quadratic curve, whenever the angle is zero or very close to zero.

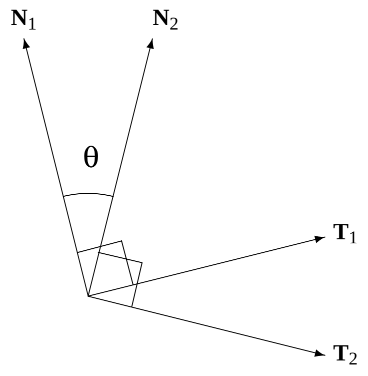


Figure 2: *Construction of the tangent vectors* $\mathbf{T}_1$ *and* $\mathbf{T}_2$ *orthogonal to* $\mathbf{N}_1$ *and* $\mathbf{N}_2$ *respectively, lying in the plane spanned by* $\mathbf{N}_1$ *and* $\mathbf{N}_2$

We shall derive a curve spanned by the tangent vectors and vertex points which is as relaxed as possible. This curve can be obtained by minimizing the integral which is the total sum of the square acceleration:

$$\int_0^1 \| f''(t) \|^2 dt \qquad (3)$$

on some variable β that controls the tangent length and apply it on the boundary curve. This defines the least square acceleration. However, a second degree curve can only have one derivative determined when it is also determined to pass through two end points. Therefore, we have to find a second degree curve spanned by the tangent vectors and vertex points which have optimal fit between both these tangent vectors and the derivatives. We shall derive such a curve by finding an optimal fit for the first tangent, then we can minimize the integral in equation (3).

2.1. Optimal Fit for the First Tangent

The quadratic curve is given by the equation:

$$\mathbf{f} = \mathbf{a}t^2 + \mathbf{b}t + \mathbf{c}. \qquad (4)$$

We have the initial conditions:

$$\mathbf{f}(0) = P_1, \qquad (5)$$

$$\mathbf{f}(1) = P_2, \qquad (6)$$

$$\mathbf{f}'(0) = \alpha\beta\mathbf{T}_1, \qquad (7)$$

$$\mathbf{f}'(1) = \beta\mathbf{T}_2, \qquad (8)$$

where $\mathbf{T}_1$ and $\mathbf{T}_2$ is the tangent vectors corresponding to $\mathbf{N}_1$ and $\mathbf{N}_2$ respectively. The constants α and β are to be determined. The value of α will determine optimal (in the least square-sense) fit between the curve and the second tangent. The value of β will determine the least square acceleration of the curve in the interval [0,1]. It can be shown that:

$$\mathbf{c} = P_1, \tag{9}$$
$$\mathbf{a}+\mathbf{b}+\mathbf{c} = P_2, \tag{10}$$
$$\mathbf{b} = \alpha\beta\mathbf{T}_1. \tag{11}$$

Let $\mathbf{P}$ be the vector from P_1 to P_2:

$$\mathbf{P} = P_2 - P_1. \tag{12}$$

Then we have:

$$\mathbf{a} = \mathbf{P} - \alpha\beta\mathbf{T}_1. \tag{13}$$

At the first edge, we have the initial condition that the tangent is equal to the derivative, as shown in equation (7). Furthermore, we want to determine α so that the difference between the derivative and the tangent is as small as possible. Thus:

$$\frac{\partial}{\partial\alpha}\left\{ \|\mathbf{f}'(1) - \beta\mathbf{T}_2\|^2 \right\} = 0. \tag{14}$$

In this way we seek the least-square difference between the tangent and the derivative. The difference of the curve slope and the polygon tangents in one of the ends equals zero, but at the other end minimization can be performed:

$$\frac{\partial}{\partial\alpha}\left\{ (2\mathbf{a}+\mathbf{b}-\beta\mathbf{T}_2)^2 \right\} = 0 \Rightarrow \tag{15}$$
$$2\alpha\beta^2\mathbf{T}_1\cdot\mathbf{T}_1 - 4\beta\mathbf{P}\cdot\mathbf{T}_1 + 2\beta^2\mathbf{T}_1\cdot\mathbf{T}_2 = 0. \tag{16}$$

Finally, divide both sides of the equation (16) by -2β and rearrange the terms:

$$2\mathbf{P}\cdot\mathbf{T}_1 - \alpha\beta\mathbf{T}_1\cdot\mathbf{T}_1 - \beta\mathbf{T}_1\cdot\mathbf{T}_2 = 0. \tag{17}$$

In order to find an α that fulfills the criterion set in equation (17) we must also determine β.

2.2. Finding the Least Square Acceleration

Next, we determine β such that the integral of the acceleration is least-square minimum in the interval [0,1]:

$$\frac{\partial}{\partial\beta}\int_0^1 \|\mathbf{f}''(t)\|^2 dt = 0 \tag{18}$$

The second derivative of $\mathbf{f}(t)$ is constant: $\mathbf{f}'' = 2\mathbf{a}$. Since it does not depend on the variable t we have:

$$\frac{\partial}{\partial\beta}\|\mathbf{2}a\|^2 = 0 \Rightarrow \tag{19}$$
$$8\alpha^2\beta\mathbf{T}_1\cdot\mathbf{T}_1 - 8\alpha\mathbf{P}\cdot\mathbf{T}_1 = 0. \tag{20}$$

Divide both sides of the equation (20) by -8β and rearrange the terms:

$$\mathbf{P}\cdot\mathbf{T}_1 - \alpha\beta\mathbf{T}_1\cdot\mathbf{T}_1 = 0. \tag{21}$$

This gives us a system with the two equations 17 and 21. The solution becomes:

$$\alpha = \frac{\mathbf{T}_1\cdot\mathbf{T}_2}{\mathbf{T}_1\cdot\mathbf{T}_1}, \tag{22}$$
$$\beta = \frac{\mathbf{P}\cdot\mathbf{T}_1}{\mathbf{T}_1\cdot\mathbf{T}_2}. \tag{23}$$

This is valid for the quadratic curve

$$\mathbf{f}_1(t) = (\mathbf{P} - \alpha\beta\mathbf{T}_1)t^2 + \alpha\beta\mathbf{T}_1 t + P_1. \tag{24}$$

2.3. Near least Square Acceleration

So far we have a curve where the derivative of $\mathbf{f}_1(t)$ at P_1 is the same as the tangent $\mathbf{T}_1$ and the derivative at P_2 is as close to $\mathbf{T}_2$ as the least square minimization allows. If we had chosen to have the same derivative of the function as the tangent at P_2 and optimize at P_1 instead, we would get another curve. Which one will be the best? It is hard to tell. However, we could get an approximation by taking the average of both. Hence, we will get an near least square acceleration second degree curve which is close to optimal in both ends.

2.4. Same Procedure for the Other Side

The next step is to repeat the same procedure for the other tangent $\mathbf{T}_2$ at the other end of the curve, in order to obtain $\mathbf{f}_2(t)$. We use the initial conditions:

$$\mathbf{f}(0) = P_1, \tag{25}$$
$$\mathbf{f}(1) = P_2, \tag{26}$$
$$\mathbf{f}'(0) = \beta'\mathbf{T}_1, \tag{27}$$
$$\mathbf{f}'(1) = \alpha'\beta'\mathbf{T}_2, \tag{28}$$

Then we derive the boundary curve using these conditions, where:

$$\mathbf{c} = P_1, \tag{29}$$
$$\mathbf{a}+\mathbf{b}+\mathbf{c} = P_2, \tag{30}$$
$$2\mathbf{a}+\mathbf{b} = \alpha'\beta'\mathbf{T}_2. \tag{31}$$

This system of equations gives:

$$\mathbf{a}+\mathbf{b} = P_2 - P_1 = \mathbf{P}, \tag{32}$$
$$\mathbf{a} = \alpha'\beta'\mathbf{T}_2 - \mathbf{P}, \tag{33}$$
$$\mathbf{b} = 2\mathbf{P} - \alpha'\beta'\mathbf{T}_2. \tag{34}$$

Finally, the quadratic curve becomes:

$$\begin{aligned}\mathbf{f}_2(t) = &(\alpha'\beta'\mathbf{T}_2 - \mathbf{P})t^2 + \\ &(2\mathbf{P} - \alpha'\beta'\mathbf{T}_2)t + P_1.\end{aligned} \tag{35}$$

Repeating the minimization process of $\mathbf{f}_2(t)$ for both β and α gives:

$$\alpha' = \frac{\mathbf{T}_1 \cdot \mathbf{T}_2}{\mathbf{T}_2 \cdot \mathbf{T}_2}, \tag{36}$$

$$\beta' = \frac{\mathbf{P} \cdot \mathbf{T}_2}{\mathbf{T}_1 \cdot \mathbf{T}_2}. \tag{37}$$

2.5. Putting it all together

By taking the mean value of (24) and (35) we get a symmetric curve with near least square acceleration:

$$\mathbf{f}_3(t) = \left(\frac{\alpha'\beta'\mathbf{T}_2 - \alpha\beta\mathbf{T}_1}{2}\right)t^2 + \left(\mathbf{P} + \frac{\alpha\beta\mathbf{T}_1 - \alpha'\beta'\mathbf{T}_2}{2}\right)t + P_1. \tag{38}$$

It is easy to prove that it will be symmetric. The derivative is:

$$\mathbf{f}_3'(t) = (\alpha'\beta'\mathbf{T}_2 - \alpha\beta\mathbf{T}_1)t + \left(\mathbf{P} + \frac{\alpha\beta\mathbf{T}_1 - \alpha'\beta'\mathbf{T}_2}{2}\right). \tag{39}$$

Moreover, the derivatives at the edges are:

$$\mathbf{f}_3'(0) = \mathbf{P} + \frac{\alpha\beta\mathbf{T}_1 - \alpha'\beta'\mathbf{T}_2}{2}, \tag{40}$$

$$\mathbf{f}_3'(1) = \mathbf{P} - \frac{\alpha\beta\mathbf{T}_1 + \alpha'\beta'\mathbf{T}_2}{2}. \tag{41}$$

Clearly, the derivatives will be symmetric around $\mathbf{P}$ which is the vector between the vertex points. Since the curve is quadratic we can not get a curve that has derivatives equal to the tangents at the vertex points. However, for a quadratic curve, this is as close as we can get with the least square acceleration requirement.

2.6. Invariance with respect to Normalization

We stated earlier that the tangent vectors do not need to be normalized and we shall prove that this is true. Expand the terms $\alpha\beta\mathbf{T}_1$ and $\alpha'\beta'\mathbf{T}_2$:

$$\alpha\beta\mathbf{T}_1 = \frac{\mathbf{T}_1 \cdot \mathbf{T}_2}{\mathbf{T}_1 \cdot \mathbf{T}_1}\frac{\mathbf{P} \cdot \mathbf{T}_1}{\mathbf{T}_1 \cdot \mathbf{T}_2}\mathbf{T}_1 = \frac{\mathbf{P} \cdot \mathbf{T}_1}{\mathbf{T}_1 \cdot \mathbf{T}_1}\mathbf{T}_1, \tag{42}$$

$$\alpha'\beta'\mathbf{T}_2 = \frac{\mathbf{T}_1 \cdot \mathbf{T}_2}{\mathbf{T}_2 \cdot \mathbf{T}_2}\frac{\mathbf{P} \cdot \mathbf{T}_2}{\mathbf{T}_1 \cdot \mathbf{T}_2}\mathbf{T}_2 = \frac{\mathbf{P} \cdot \mathbf{T}_2}{\mathbf{T}_2 \cdot \mathbf{T}_2}\mathbf{T}_2. \tag{43}$$

Remember that $\mathbf{P}$ is an vector and not a point and therefore equation (42) is the projection of $\mathbf{P}$ on $\mathbf{T}_1$. Likewise, equation (43) is the projection of $\mathbf{P}$ on $\mathbf{T}_2$. Hence, normalization of $\mathbf{T}_1$ and $\mathbf{T}_2$ is not necessary, since projection of one vector onto another is independent of the other vectors length.

3. Quadratic curvilinear surfaces

A quadratic surface can be determined by using the six control points[8] in figure 3. The edge mid points $\mathbf{P}_{12}, \mathbf{P}_{23}$ and

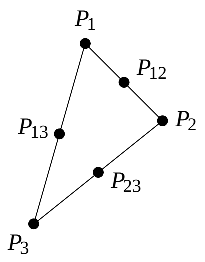


Figure 3: *Six control points*

$\mathbf{P}_{13}$ can be determined by setting $t = 0.5$ in (38). By using a quadratic surface, it is possible to use any kind of subdivision scheme without recalculating new points in a recursive manner. Instead, new points are retrieved from the quadratic surface.

4. Rendering the Curvilinear Polygons

Figure 4 shows a wire frame rendering of a Torus. It is quite obvious that the Torus has got straight edges on the outer contour. These can be made smooth by using the proposed method. Figure 5 shows the same Torus, but this time, the curvilinear mesh is used. The contour is now smooth.

To our knowledge, most modern hardware do not have the capability to insert new polygons on the fly. The only possible solution, in order to make the contour smooth, is to subdivide the whole object before rendering. However, the result is a more complex object. Moreover, it will take more time to render it since it has more polygons. It would be preferable if we could subdivide the object on the fly, only where extra polygons are needed, i.e on the contour. The proposed method could be used to produce these extra polygons that will make the object smoother.

The extra points will lie on the quadratic surface, that will have near least square acceleration. This will assure that the contour will be as relaxed as possible using quadratic boundary curves. However, since quadratic surfaces are used, C^1 continuity is not guaranteed. If the angle between the normals is large, then the constructed curve can make a rather large bend. This implies that there is a sharp edge between the polygons or that the polygons should have been subdivided prior rendering. This is of course a problem common also for shading, not only for this method.

However, subdivision on the fly is not as easy as one might think at first glance. First of all, we must be able to determine which polygons lie on the contour. This can be done if pointers to neighboring polygons are available. Then, it is possible to check if a neighboring polygon is back facing. Secondly, it is possible that a polygon edge that lies behind another polygon can have a large curvature and if subdivision is performed, then it will be visible. As long as it is considered being hidden, it is no problem but when it eventually lies on the contour, due to that the object is rotating, it will suddenly pop up as a heavily curved contour. This problem is generally known as popping.

We suggest these problems for further research. Nonetheless, the proposed method could be used for determining new points for any subdivision scheme that will subdivide an object in order to make it appear more smooth.

Vlachos et al.[9] suggest that their PN triangles should be subdivided on the graphics chip without any software assistance. The proposed method could also be used for hardware subdivision since it is based on the available vertex information for each polygon. Furthermore, it should be an attractive method for hardware implementation, since the bottleneck in todays graphics hardware is in feeding the graphics processor with triangles at a sufficient rate. Thus, fewer triangles could be transfered and more triangles are created on the fly on the graphics chip. This can not easily be done for the contour only, since information from neighboring polygons are needed. Instead, on the fly subdivision is done for the whole object.

Another interesting fact is that the dot product of **P** and the tangents $\mathbf{T}_1$ and $\mathbf{T}_2$ is invariant under rotation and translation. Hence, these products could be precomputed. This is probably better for software rendering where it is possible to save computation time. For a hardware implementation, this means that two new variables must be transferred through the graphics pipeline and this will make the bottleneck problem previously discusses even worse.

5. Conclusions

It has been shown that a curvilinear mesh with near least square acceleration for quadratic surfaces could be constructed from vertex normals and vertex points. This mesh can then be subdivided using an appropriate subdivision algorithm. The proposed method will be fast, since the computed tangents does not need to be normalized. Furthermore, it has been shown that some terms can be precomputed in order to speed up software rendering.

5.1. Future Work

We propose for future work, how the presented method for constructing a curvilinear mesh, could be used in order to make on the fly subdivision for contours only. Moreover, it should be easy to use the method for on the fly subdivision on the graphics chip. This would decrease the problem of the bandwidth bottleneck.

It should also be determined if this method is good enough for more complex objects, or if cubic surfaces are preferred, even though they are computationally more expensive. Furthermore, it should be investigated whether the average function (38) could be replaced with either of the functions (24)

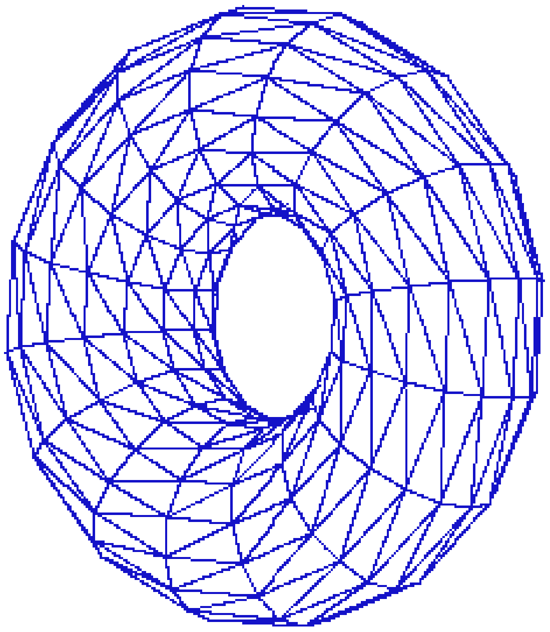


Figure 4: *A wire frame rendering of a Torus*

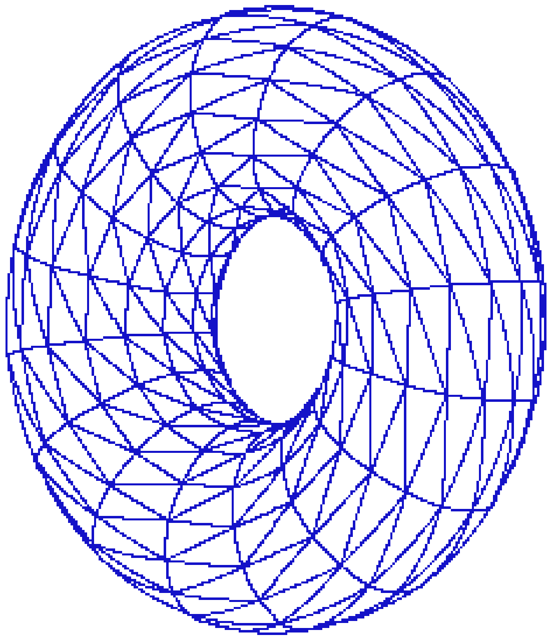


Figure 5: *A curvilinear mesh for a Torus*

and (35) depending on how the normals are pointing compared to the edge.

References

1. J. Bruijn. Quadratic Bezier triangles as drawing primitives *Proceedings of the 1998 EUROGRAPHICS/SIGGRAPH workshop on Graphics hardware 1998 , Lisbon, Portugal* pp. 15 - 24, 1998
2. E. Catmull, J. Clark. Recursively generated B-spline Surfaces on arbitrary Topological Meshes. *Computer Aided Design*, 10 pp.350 - 355, Oct 1978.
3. Leif Kobbelt. $\sqrt{3}$-subdivision *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pp. 103 - 112, July 2000.
4. Charles Loop. Smooth subdivision surfaces based on triangles. *Master s thesis, University of Utah, Department of Mathematics*, 1987.
5. J. Maillot, J. Stam. A unified Subdivision Scheme for Polygonal Modeling *Proceedings Eurographics 2001*, Vol 20, No 3, pp. 471-479, 2001.
6. W. K. Nicholson. Linear Algebra With Applications *PWS Publishing Company*, Third Edition, pp. 275, 1995.
7. K. van Overveld and B. Wyvill. An Algorithm for Polygon Subdivision Based on Vertex Normals *Proceedings Computer Graphics International 1997*, pp. 3-12, 1997.
8. L. Seiler. Quadratic Interpolation for Near-Phong Quality Shading *Proceedings of the conference on SIGGRAPH 98: conference abstracts and applications*, Page 268, 1998.
9. A. Vlachos , J. Peters , C. Boyd , J. L. Mitchell. Curved PN triangles *Proceedings on 2001 Symposium on Interactive 3D graphics, March 2001* pp. 159-166, 2001.
10. P. Volino, N. Magnenat-Thalmann. The SPHERIGON: A Simple Polygon Patch for Smoothing Quickly your Polygonal Meshes. *Computer Animation'98, proceedings, IEEE Press*, pp.72-79, 1998
11. A. Watt. 3D Computer Graphics. *Addison-Wesley*, pp.59-64, 2000.

Time Machine Oulu - Multichannel Augmented Reality

Jaakko Peltonen
Oulu, Finland
jaakko.peltonen@oulu.fi

Oulu is a city of technology, technopolis as they say, but only small number of the inhabitants know even some facts about its history. We have implemented a ‘time-machine’, that can represent the city in terms of past physical structures. A new dimension is also added, a database of Oulu containing not only buildings, but also all kinds of information and activities that has happened the city over the years such as fires. The material for the database was obtained from a fire-insurance company named Tarmo Oy. It was insuring most of the buildings in Oulu, and had material representing approximately 80% of the city. The insurance documents have been verified to be accurate. The characteristics for the city of Oulu in 18th and 19th century is quite unique for the region. The city was fairly large and major part of the houses were built from timber logs. The city was a vast number of small wooden 1 and 1 ½ story height buildings. The current database covers the years between two devastating fires in Oulu, 1822-1882. There are over 3500 buildings with more than 110 records at the most in the database. Soon the years 1882 to 1950, which is approximately the same time scale as the previous time period will be merged into the current database.

The interesting aspect to Time Machine Oulu is that it can provide different views according to the type of user. A researcher receives only the exact, academic view. That is, data that has been correctly validated. Researchers can update the information and add new buildings as well as other material. They can also access the original data which could be an original insurance map. Normal users receive views of the plain model, and are provided with less research orientated data. The other type of user is one that is interested in ‘edutainment’. The user receives narrative descriptions to the levels and time periods, which can be used with their own imagination. Each of these different types of users demand robustness in the system and understanding of the type of client the user is using. The current system allows for browsing over the Web through an VRML compliant browser. We have had successful results using the system with a wireless LAN and a Pocket PC. In this sense, Time Machine Oulu is experimenting with what we term, Multichannel Augmented Reality, through the use of multiple different client types for examining the information in the database, as well as the possibility of been immersed in the actual world, and seeing a bygone era. The idea for using a 3D model came almost instantly when the material was started to gather. The main idea was to model a block of buildings as a stationary model, and create some still renderings and some high quality animations. It soon became apparent that the scope of the material was large and the shape of the city started to form out. Since the material and the data
for each building was already in database, a natural extension was to model the city in 3D based on SQL queries. VRML was chosen and in combination with a backend architecture of Apache, mySQL and PHP4. The first step was taken to enlarge the visual representation of historical town. The basic 3D model represents merely dimensions of object, materials, lights, animation objects and scripting are further levels.

Sharing the database to the Internet is somewhat straightforward procedure. Using VRML makes the site more dependent on the client connections, browser software, and ability to handle different constraints demanded by the client. Parameters include level of detail in the model, textures, other model features such as animations, client operating system, client hardware type, network throughput and user settings. VRML-generation for each client is different and the system is evolving constantly. Future work includes location based services where users can walk through Oulu, and visualize with their PDA, the environment in any time period. This will mean examining more issues around model optimizations, bandwidth and hybrid rendering algorithms.

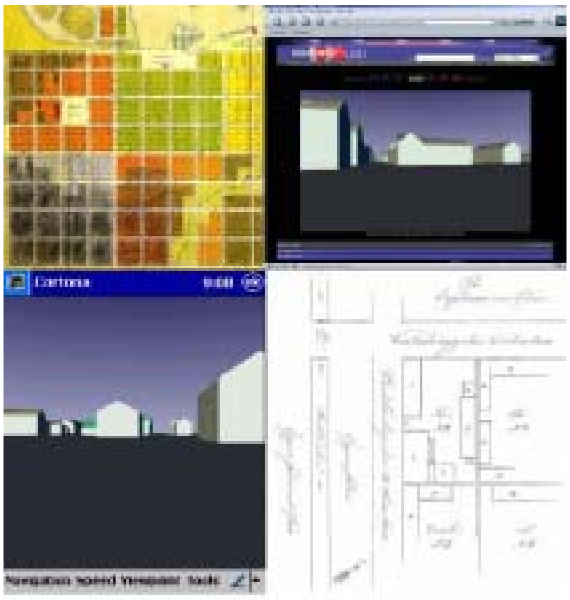


Figure: From top left moving clockwise. 1. Original insurance map of city. 2. PC based browser of VRML world generated from SQL query. 3. Original drawing from fire insurance register of a block. 4. Pocket PC browser of world over wireless LAN.For more information, visit the MediaTeam homepage on www.mediateam.oulu.fi