Abstract

I present a general overview of research in texture-based techniques for rendering complex virtual environments that is being conducted by members of the Walkthrough Project at The University of North Carolina at Chapel Hill. I then describe in detail a specific implementation of a texture-based approach to rendering global illumination. Polygonal meshes that are produced by a radiosity process are captured as texture maps in order to dramatically reduce polygon counts and improve rendering performance. This kind of technique takes advantage of current hardware trends to provide fast texturing capabilities. I argue that this approach can be applied to a wide range of difficult rendering situations and will allow us to render increasingly complex virtual environments at interactive speeds.

1 Introduction

As long as there have been computer graphics, people have been inspired by the possibility that some day we may be able to create artificial environments that are as vivid and complex as reality itself. Members of the Walkthrough project at UNC conduct research in a wide range of areas related to the creation and display of very large and complex virtual environments. The focus of current research is on real world problems such as architectural environments that contain up to tens of millions of polygons.

I will begin with a brief overview of the Walkthrough project, with a focus on current research in texture based rendering techniques. I then describe ongoing work with illumination-as-textures (IAT), and present a detailed explanation of the current software implementation of this strategy. Finally, I describe future work in the area of IAT and speculate about possible research directions and opportunities.

This document is intended for faculty, staff, and students in the Computer Science Department who are interested in learning about the implementation of IAT, as well as current research in texture-based methods by the Walkthrough project. I assume that the reader has some knowledge of texture mapping techniques, a basic understanding of the performance demands for a virtual environment, and a rudimentary knowledge of graphics hardware, graphics rendering, and radiosity.

2 Texture Based Research and the Walkthrough Project

The stated goals of Walkthrough are to create virtual environments that are "faster, prettier, handier, and realer." The group focuses on real world problems and seeks solutions that use commercially-available hardware and software as much as possible. Because of the current trend in graphics hardware to provide fast texture-mapping capabilities, a major topic of current research for Walkthrough is on texture based approaches to rendering virtual environments.

John Poulton of the PixelFlow research group has pointed out another motivation for pursuing texture or image based approaches to rendering. He claims that the trend in graphics hardware will soon support scenes with tens of millions of polygons, and as a result, the size of the average polygon on the screen will shrink to a single pixel. Because of this, it no longer makes sense to render polygons as three-dimensional entitities, and we should instead focus on rendering pixels or images.

Walkthrough uses texture based techniques to render elaborate surface detail as well as areas or objects of significant geometrical complexity within an environment. The three major areas that we are currently investigating are standard texture, geometry-based texture, and illumination-as-texture.

2.1 Standard Texture

The Walkthrough project is primarily concerned with simulations of architectural environments. This type of environment often has a large number of surfaces that benefit greatly in appearance from surface detail textures. Whether it is wood grain or bathroom tile, these textures lend a much greater appearance of realism than simple shaded polygons.

2.2 Geometry Based Texture

Geometry based texture (GBT), is a relatively new approach to rendering. This technique smoothly replaces complex geometry with a view-dependent texture map based on distance from the viewpoint, direction the viewer is facing, or a stress-based criteria. Texture maps thereby serve as the lowest level of detail for objects in the environment.

In addition, vistas through doorways or windows can be *capped* with a view dependent texture. Instead of rendering geometry that is visible through a window, for example, a polygon is rendered in the opening that bears a texture map of the geometrical scene that would be visible through the window. To improve the realism of the effect, the texture map is transformed dynamically according to the viewing angle. In this way, we can simulate three-dimensional behavior with several two-dimensional texture maps.

2.3 IAT

IAT is a technique by which global illumination for a model is captured and displayed in the form of texture-mapped surfaces. This area of research and a specific implementation of IAT are the primary topics of this paper.

I begin with an overview of the original system that was designed and written at UNC. This description should allow someone to understand the workings of the system and the code well enough to both use the system and extend its functionality. Next, I present the results in terms of both performance and memory use shown by models processed by the system. I compare the performance of models processed with IAT to the original polygon based radiosity version. Finally, I describe improvements and extensions to the system that are currently underway.

3 Our Original Implementation of IAT

I developed our initial software implementation of IAT during the Spring semester of 1995. Work was sponsored by the Walkthrough project and was also submitted as a final project for COMP 239, "Exploring Virtual Worlds." I continued working on the system during the following Summer while employed at Silicon Graphics Incorporated in Mountain View, California.

I describe the current implementation of IAT in terms of seven major topics related to the process. The topics are discussed in decreasing order of their generality. I begin with a description of the most general concepts; the model pipeline and model representation. I follow this with a more specific explanation of the generation and display of illumination textures. Finally, I end with the specific details of the implementation that were dictated by performance or hardware concerns; resource management, texture scaling, and texture combination.

3.1 The Model Pipeline

The model pipeline is the pathway by which a model is created, converted, radiositized, textured and converted again, and finally displayed. Several of these stages influenced the design of the IAT implementation. For example, I decided to embed the IAT process in a conversion stage, because the necessary data structures were already being traversed for the conversion at that point.

Transitions between the various stages in the model pipeline are made using different invocations of a tool called the Translator. This is a software system that is currently under development for Walkthrough that can read, write, and convert to and from a variety of model file formats. The Translator also provides facilities for model processing, such as IAT, as well as other operations and model cleanup.

3.1.1 MultiGen

The first stage of the model pipeline is model creation. Models are created using MultiGen modeling software available in the Graphics Lab or are obtained by some other means. The Translator currently supports MultiGen Openflight 14.2 (.flt), AutoCad (.dxf), Pixel Planes (.pphigs), Lightscape preparation file format (.lp), and Lightscape solution file format (.lsb), where the parenthesized expressions indicate conventional file name extensions.

3.1.2 Lightscape

In the second stage of the pipeline, the model is converted into a Lightscape preparation file format (.lp) using the Translator. The preparation file can then be loaded into the Lightscape radiosity tool from Lightscape Technologies. Using this radiosity tool, the user can interactively add and position lights in the model. A radiosity solution for the model is then computed and saved out as a Lightscape solution file (.lsb).

3.1.3 IAT

In the third stage, the translator reads the Lightscape solution file and generates a texture mapped version of the model. The user can select a criteria which determines the number and size of texture maps that are generated. The output from this conversion is an Openflight 14.2 file (.flt).

3.1.4 Perfjoy

For the final stage of the model pipeline, Walkthrough uses an application called "perfjoy" to visualize and navigate models on a Silicon Graphics Onyx Reality Engine 2. Perfjoy is based on IRIS Performer, and can load and display the Openflight file generated from the IAT stage of the model pipeline. Perfjoy can also be used to generate performance statistics for a given model, such as rendering speed, total polygon count, and other useful data.

3.2 Model Representation

The representation of models is important to the various stages of the model pipeline. Some stages of the pipeline need to be run on a particular platform, while other stages are architecture-independent. In addition, the various model file formats that are used in the pipeline affect the types of data that the models can contain.

Although both MultiGen and Lightscape must be run on a Silicon Graphics machine, the Translator and the code that generates illumination textures can run on Silicon Graphics or Hewlett Packard and will soon be ported to Sun workstations. The ability to run the translator on multiple platforms is important because converting a very large model can sometimes take several minutes, so it is advantageous to be able to run simultaneous model conversions on many different machines in the Graphics Lab. The same is true for the IAT code.

The various file formats used in the pipeline have both advantages and disadvantages. Openflight 14.2 is extensive and can hold a large amount of model data for both modeling and rendering purposes. Openflight is a binary format, which makes it compact in memory. Because Openflight is primarily designed for modeling purposes, it is not optimized for fast rendering. Perfjoy converts Openflight files into Performer run-time format in order to render the model more quickly during a simulation.

Lightscape format is primarily designed to facilitate the computation of global illumination. This format contains a large amount of data concerning surface characteristics and other lighting properties. Lightscape format does not preserve object-level hierarchy all the way to the solution stage, however, and is not optimized for rapid display. Perfjoy can convert a Lightscape model directly into Performer run-time format for more efficient display.

3.3 Generation of Illumination Texture

Illumination textures are generated during the Illumination As Texture stage of the model pipeline. IAT software takes advantage of data structures in the Lightscape file that are well suited for producing texture maps. I will later claim that the Openflight file that is created is equal or better in appearance than the original polygonal version of the radiosity solution.

Lightscape computes a radiosity solution for an input model by adaptively subdividing the various surfaces within the database. Surfaces are subdivided according to discontinuities in the color values across that surface resulting from lights within the environment. Surfaces with discontinuous lighting information, shadows, and highlights tend to be subdivided many more times than surfaces that receive a more uniform distribution of light.

The output of the radiosity process is a model where many of the original polygonal surfaces have been subdivided into polygonal meshes (see Figure 1). These meshes contain lighting information for the surface in the form of color at every vertex in the mesh. The Lightscape format stores this mesh information as a quad-tree, where the original polygon is the root and its children are the result of recursive subdivision. The roots of each mesh are guaranteed to be either a quadrilateral or a triangle.

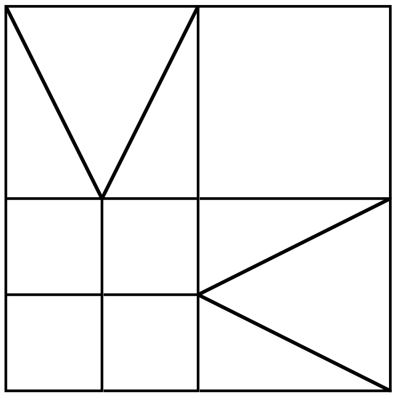

Figure 1: Typical polygonal mesh resulting from adaptive subdivision for radiosity

The quad-tree representation of the mesh turns out to be extremely useful for producing texture maps that capture the same color information. Notice that a full quad tree can be *flattened* into a square grid, where each position in the grid corresponds to a leaf of the tree (see Figure 2). We can easily generate a texture map from a quad tree by filling in texels with the color values contained at their corresponding leaves. When the quad tree is not full, as is often the case with radiosity solutions, we simply use bilinear interpolation to fill in the missing values in the texture map.

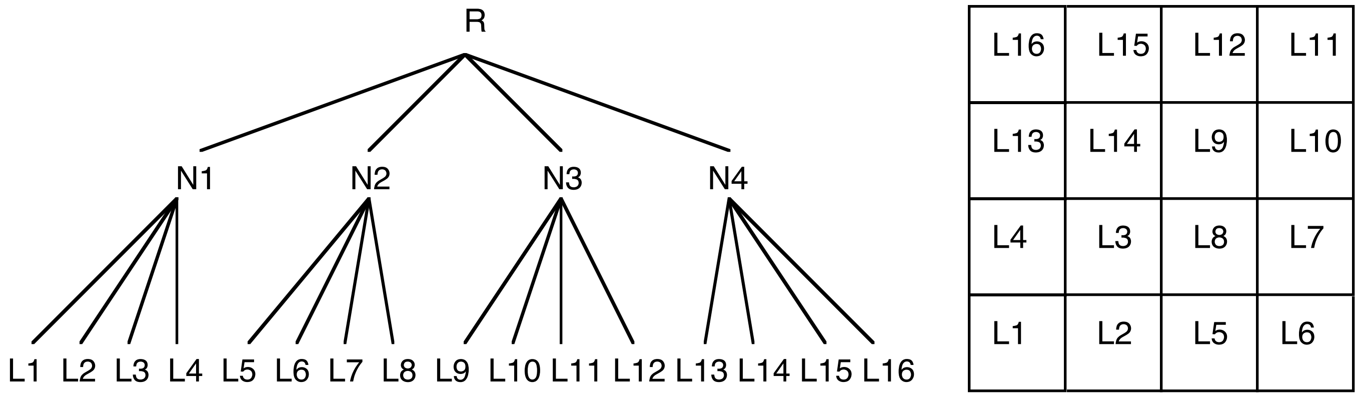

Figure 2: Full quad tree and its flattened representation as a square grid

Texture maps produced from the quad tree in this way capture the same information as the original polygonal mesh. Every vertex in the mesh is replaced by a texel in the texture map. The size of the texture is therefore directly related to the discontinuity of the illumination across the original surface. Because we generate textures that are just large enough to represent the radiosity data, we make efficient use of expensive texture memory. The appearance of the final textured surface is equivalent to the polygonal version because both representations contain and display the same color information at the same locations on the surface polygon. Texture hardware on the SGI bilinearly interpolates between texels in the texture map, giving the surface the same smooth appearance of the polygonal version. Because the SGIs also support hardware mip-mapping and texture perspective correction, these textured surfaces remain stable throughout all transformations that tend to occur during simulations of a model.

3.4 Application of Illumination Textures

Illumination textures are automatically applied to the appropriate surfaces within the model. The pre-radiosity polygons can be reproduced by simply reading the root node of the quad tree for each surface while ignoring the rest of the tree. Textures are oriented correctly on these polygons and shifted beyond the extents of the polygon by one half of one texel to prevent errors in texture map interpolation during display.

3.5 Resource Management

The user can interactively balance the number of textures versus polygons in the final model. The IAT process uses a threshold value to determine if a texture map is generated for a particular surface. Otherwise, the surface is output as a polygonal mesh. The threshold value can be set by the user and represents the depth of the mesh or the number of times the particular surface was subdivided. If the threshold is set at zero, for example, an illumination texture map will be generated for every polygon in the input model. If the threshold is set at 2, only polygons that were subdivided two or more times as a result of the radiosity calculation will be texture mapped. In this way, we avoid generating very small textures that are more expensive to render than the fairly trivial polygonal meshes they would replace.

In the current implementation of IAT, the threshold value is determined in an ad hoc fashion for each model. The amount of available texture memory, as well as the performance of the model in a simulation are important factors when deciding on a threshold value. A surface that has been adaptively subdivided n times will result in a mesh that contains between (4 + 7 * (n - 1)) and (4 ^ n) polygons. For example, a mesh resulting from 3 levels of regular subdivision will have 4 ^ 3 = 64 polygons, or it can be replaced by a single polygon with a texture map. The decision to replace this mesh with a texture map will depend on the amount of texture memory available. Myszkowski [2] proposes an empirical approach to determine when a mesh should be replaced with a texture map.

3.6 Texture Scaling

It is sometimes necessary to reduce the amount of texture memory that is used by a means other than thresholding. For very large simulations or for simulations on platforms that have limited texture memory, illumination textures can be scaled after they are generated. This will often produce visual artifacts such as blurring or correspondence errors between neighboring surfaces.

3.7 Texture Combination

Some hardware platforms permit the combination of two texture maps on the same surface in a single rendering pass. This allows us to combine illumination textures and surface detail textures on the fly. On such a platform it is possible to render a brick wall with shadows and other non-uniform illumination. The brick texture may be repeated on the face and combined with a low-resolution illumination map that is applied with a scale of 1.0 to the face. This results in the appearance of a high-resolution surface image from fairly low resolution texture maps. Surface detail textures, such as the brick pattern discussed, are applied in the original model and preserved throughout the modeling pipeline.

4 Performance of IAT Models

The success of the system can be evaluated in terms of appearance, performance, and memory requirements. The immediate goal of the project was to produce a model that looked as good as the original, but could be rendered significantly faster. The idea was then to refine the process by improving the appearance and making more efficient use of both texture and main memory.

4.1 Appearance

Models with illumination textures appear to be identical to the polygonal versions. All the data in each radiosity mesh is captured fully by a texture map. The texture maps are bilinearly interpolated in hardware, which produces the same appearance as the polygons of the mesh version that have a color at every vertex.

4.2 Speed

Models that are processed using my implementation of the illumination as texture approach show significantly better performance in virtual environment simulations (see Figure 3). Performance statistics were obtained by running a basic Performer application on a Silicon Graphics Onyx Reality Engine 2 with 16 megabytes of texture memory. Note both frames per second and memory usage. The textured version takes longer to load because the texture maps must be downloaded from main memory into texture memory, where they are then packed efficiently according to size.

Name	**No. Triangles**	**Frame Rate** (frames/second)	**Texture Memory** (megabytes)
maus (polygon)	95536	7.5	-
maus (IAT)	39820	15.0	1.26
cave (polygon)	43960	20.0	-
cave (IAT)	19492	30.0	0.77
lavapit (polygon)	55158	8.6	-
lavapit (IAT)	7183	30.0	1.48
pool (polygon)	84384	8.6	-
pool (IAT)	39865	15.0	2.76

Figure 3: Performance of IAT models compared to the polygonal versions

IAT models exhibit faster rendering because of the tradeoff between transformations and rasterization. For an IAT model, a single polygon is transformed according to the viewing matrix for each surface in the scene, compared to a transformation for every polygon in the mesh at each surface in the polygonal radiosity version of the model. A large polygon must be rasterized for every surface in an IAT model, whereas many smaller polygons are rasterized at every surface in the polygonal radiosity version. For models with significant mesh complexity (i.e. a tenfold increase in the original polygon count as a result of the radiosity process), the transformation time for the many polygons dominates the rendering cost and the IAT version will run faster.

4.3 Space

Models with illumination texture require less RAM and more texture memory at run time. They tend to require roughly the same amount of disk storage space as the polygonal version because of the large number of image files that are associated with the model file. Texture memory can be conserved by using some of the resource management techniques discussed previously, such as thresholding.

5 Current Work on the IAT System

Members of the Walkthrough project are currently working on improvements and extensions to the original IAT implementation. These changes are designed to improve both the appearance of the model as well as the performance. Concurrently, we are attempting to improve the efficiency of texture memory use.

5.1 New Texture Generation Process

The original program generated texture maps by sampling the quad-tree for every position in the image grid. We have recently implemented a new approach that uses a polygon scan conversion algorithm to fill in the texture values. This is more efficient because it requires only a single pass through the quad tree data structure for each surface. In addition, this approach is better suited to handle difficult degenerate cases such as triangles with very obtuse or acute angles.

5.2 Better Approximation of Illumination Function

The current implementation uses simple bilinear interpolation to fill in gaps in the radiosity mesh when a texture map is being produced. We can fit a higher order polynomial function to the values in the radiosity mesh to obtain a richer approximation of the illumination function for the surface. This polynomial would allow us to use bicubic, rather than bilinear interpolation, to generate a texture map.

A polynomial approximation of the surface illumination function provides two significant enhancements to the IAT process. First, the textures generated will be smoother because of the bicubic interpolation. Second, we can easily produce textures at arbitrary resolution from the approximating function.

Bicubic interpolation should improve the appearance of the IAT texture maps. The textures will look smoother and will have fewer visual artifacts because of the higher order interpolation. IAT models will no longer be identical to the original mesh version, but they will contain fewer artificial discontinuities than the simple meshed version because many of the discontinuities are a result of the adaptive subdivision process itself and not a result of global illumination. Rushmeier [3] proposes some metrics that we can use to evaluate the quality of the images produced by our polynomial approximation. Our ability to generate good approximating polynomials will improve if we perform discontinuity meshing, which I will discuss in the next section.

We can generate texture maps at arbitrary resolutions because the polynomial approximation is a continuous function. Values at the edges can be clamped to prevent the correspondence problems between neighboring surfaces that we observed with previous attempts to scale these textures. This technique will enable us to improve our resource management, as well as produce multiple levels of detail textures for each surface.

5.3 Discontinuity Meshing

Discontinuity meshing is a pre-process computation that meshes polygons in a non-regular fashion according to the geometrical relationship between lights, receivers, and occluders in the environment. For example, a diagonal shadow on a wall might cause it to be divided into three triangles defined by the edge of the shadow. We plan to implement discontinuity meshing as a pre-processing stage on a model before it is loaded into Lightscape.

Discontinuity meshing promises to improve both the appearance and performance of the models we generate. It is difficult to capture non-axially aligned shadows smoothly in Lightscape without subdividing the face to a great extent. As a result, aliasing is common in the form of blocky edges on diagonal shadows. Discontinuity meshing should reduce aliasing, and also decrease the size of the model and textures by reducing the amount of subdivision required on each face.

In addition, discontinuity meshing will allow us to generate better approximating polynomials for the surface illumination function. If we divide polygons according to discontinuities, we can be confident that there are no significant discontinuities within a particular polygon. This knowledge makes it much easier to produce a polynomial to capture all the surface illumination data for that polygon.

5.4 Multiple Pass Techniques

SGI Reality Engine platforms do not support the combination of multiple textures on the same face in a single pass. They do, however, provide a means to project one texture onto another in an additional pass. This technique can be used to combine illumination and surface detail textures at different scales as previously described. We are currently working on the implementation of this approach.

5.5 Texture Packing

The concept of texture packing will make texture memory use much more efficient. The IAT process generates a large number of variably-sized textures. The majority of these textures are considerably smaller than the optimal size for texture memory access on an SGI Reality Engine. We plan to pack many smaller texture maps into a single larger palette to increase texture memory access speed and reduce the total number of accesses.

6 Future Work

The IAT techniques illustrated in this paper provide insight into a more general category of applications. In this section, I describe some potential alternative uses of this approach, and propose some new directions in which this line of thinking might take us.

6.1 The Panel Concept

We were motivated by the desire to render virtual environments with global illumination at a high frame rate when we set out to implement IAT. The technique to replace coplanar polygonal meshes, such as the ones generated by a radiosity process, with texture maps is based on the concept of a panel. The notion of a panel as described by Dr. Fred Brooks, is that a collection of coplanar polygons can and should be treated as a single three dimensional entity that bears some two dimensional surface information, rather than as a collection of independent three dimensional pieces. Capturing illumination as a texture mapped polygon is a good embodiment of this concept because a texture map contains data in a two dimensional parameter space, yet this information can be easily applied to the surface of the polygon in three dimensions.

6.1.1 Surface Functions

The panel as texture approach can be extended to a broader domain of surface rendering applications. Simulation of reality will inevitably need to incorporate both intrinsic and extrinsic properties of the surfaces in the environment. Global illumination on a surface is just one type among many different classes of functions that provide detail related to the intrinsic properties of a surface as well as properties related to its environment.

6.1.2 The Texture Map Representation

If you think of surface detail as a function applied to a panel, it is easy to conceive of many such functions combined on the surface in some meaningful way. Our job is simplified by the image-based representation of these functions, because now the problem is reduced to combining a series of images based on some meaningful criteria. Imagine a combination of illumination function, surface detail, bump map, and reflection map on the same panel. This could be used to simulate a wooden table that is partially polished and partially rough, with shadows and moving highlights dancing across its surface.

6.2 Possible Directions

The idea of surfaces that contain information in the form of a function can be extended to cover a variety of lighting and related surface effects. These techniques also take advantage of the simplicity and speed of the texture map representation of the surface function.

6.2.1 Dynamic Functions

Another product of this line of thinking is to render dynamic or oscillating surface functions. A good example would be to calculate different radiosity solutions for different times of day for a model of a room. We could then compute a simple linear interpolation between each stage to create the illusion of a continuous change as the sun moves across the sky. There is no reason these surface functions need to be static, as long as graphics hardware allows us the appropriate access.

6.2.2 Level of Detail

A similar interpolation technique would allow us to blend between illumination or detail textures of different resolutions. Ideally, blending could be done with hardware and should not require multiple passes. This would effectively allow smooth transitions between levels of detail for the surface function.

6.2.3 Hardware Support

In order to take full advantage of this family of texture based rendering techniques, graphics programmers will need improved hardware support and access to hardware texturing capabilities. Future graphics machines should allow direct manipulation of texture memory and the frame buffer. They should provide a means to smoothly blend between two or more textures on the same surface based on interpolation techniques. In addition, they should allow the combination of multiple textures at different scales on the same face without requiring additional rendering passes. Finally, future graphics engines should provide even larger texture memories with faster access to make these methods even more effective and powerful.

7 Conclusions

The technique of rendering illumination as textured panels appears to be a small member of a much larger set of surface operations. This type of technique is appropriate considering the increasing demands for richness in virtual environments as well as the current trend towards fast texturing in graphics hardware. I believe that this type of approach promises to carry us towards the goal of richness in virtual environments.

References

1. Cohen, Michael F. and John R. Wallace. Radiosity and Realistic Image Synthesis. Harcourt Brace & Company, Publishers, Boston, 1993.

2. Myszkowski, Karol and Tosiyasu L. Kunii. "Texture Mapping as an Alternative for Meshing During Walkthrough Animation". 5th Eurographics Workshop on Rendering. Germany, June 1994.

3. Rushmeier, H. and G. Ward and C. Piatko and P. Sanders and B. Rust. "Comparing Real and Synthetic Images: Some Ideas About Metrics". 6th Eurographics Workshop on Rendering. Dublin, June 1995.