Realistic Lighting Scenarios in Mediterranean Architecture

Effective use of modeling in conjunction with creative lighting techniques to generate realistic still images

Abstract

A closed room of Mediterranean architecture was modeled to conduct realistic lighting experiments to study the requirements to achieve a variety of specific effects. After careful consideration, POV-Ray, a multi-platform freeware raytracer, was selected for its features and flexibility. Realistic lighting required a realistic model; the room was constructed to exacting dimensions to ensure a proper fit. The replication of objects was used to quickly generate the room while maintaining a high degree of accuracy and realism in design. Central features to the room are a skylight in the ceiling and a pool of water in the middle. Various lighting scenarios were conducted, experimenting with kinds of lights, their position, and their interaction with objects, especially glass and water. Further lighting scenarios were designed with atmospheric dust and flames. The result was a series of still images that illustrate the difficulty in designing a realistic lighting scenario. While realistic modeling can be extremely effective, realistic lighting is absolutely crucial to a successful raytraced image.

The team consists of Ajay Kulkarni and Charles Yoon, students in MIT 6.837 Computer Graphics.

Introduction

MIT 6.837 Computer Graphics deals with many issues related to computer graphics, from rendering a straight line on the screen to realtime 3D graphics. However, while it deals with many different issues related to the design rationale of computer graphics, we found that it did not spend a great deal of time on the creation of realistic images. Two problem sets were spent on the design of objects and scenes, but they were more concerned with modeling than anything else. An extremely accurate model of a car can be made by taking the CAD file for a car, for example, and putting them into a ray tracer. However, to make it appear realistic, it takes far more than that: it takes realistic textures and lighting. Our project focuses on the latter.

Because ray tracing focuses on the pathways of light, the correct placement, attenuation, reflection, refraction, and transmission of lights is critical to realism. This information is useful to anyone interested in creating realistic scenes or anyone interested in experimenting with lighting. For example, an interior designer may be interested in experimenting upon a different lighting technique for a large room. Studies which deal specifically with light would be able to maximizing the realism in the resulting image, giving the designer the best possible concept of what a particular scenario would look like.

While many people assume that ray tracing is largely about realistic models, the lights are the part that make it entirely visible and give it life. As a result, it is critical that everyone involved with computer graphics understands what it takes to build a scene with effective lighting.

Goals

Modeling

Because the construction of complex, realistic lighting scenarios requires a realistic model, this portion is key to the success of the rest of the project. A suitable architectural model was to be designed and constructed. Key design parameters included a structure which could be lighted in an "interesting" manner, ease of modeling, and an overall satisfying appeal.

Lighting

We were interested in using different kinds of lighting, so we intended to study every kind of lighting that our raytracer could provide. A list of lighting scenarios should be compiled and experimented upon. Examples include, lighting with point light sources, spotlights, torches, obscured lighting, and reflected lighting. Particularly of interest is lighting in conjunction with objects, such as glass, water, and atmosphere. In addition to raytracing, radiosity will be used if time constraints permit, to create more realism in the scene.

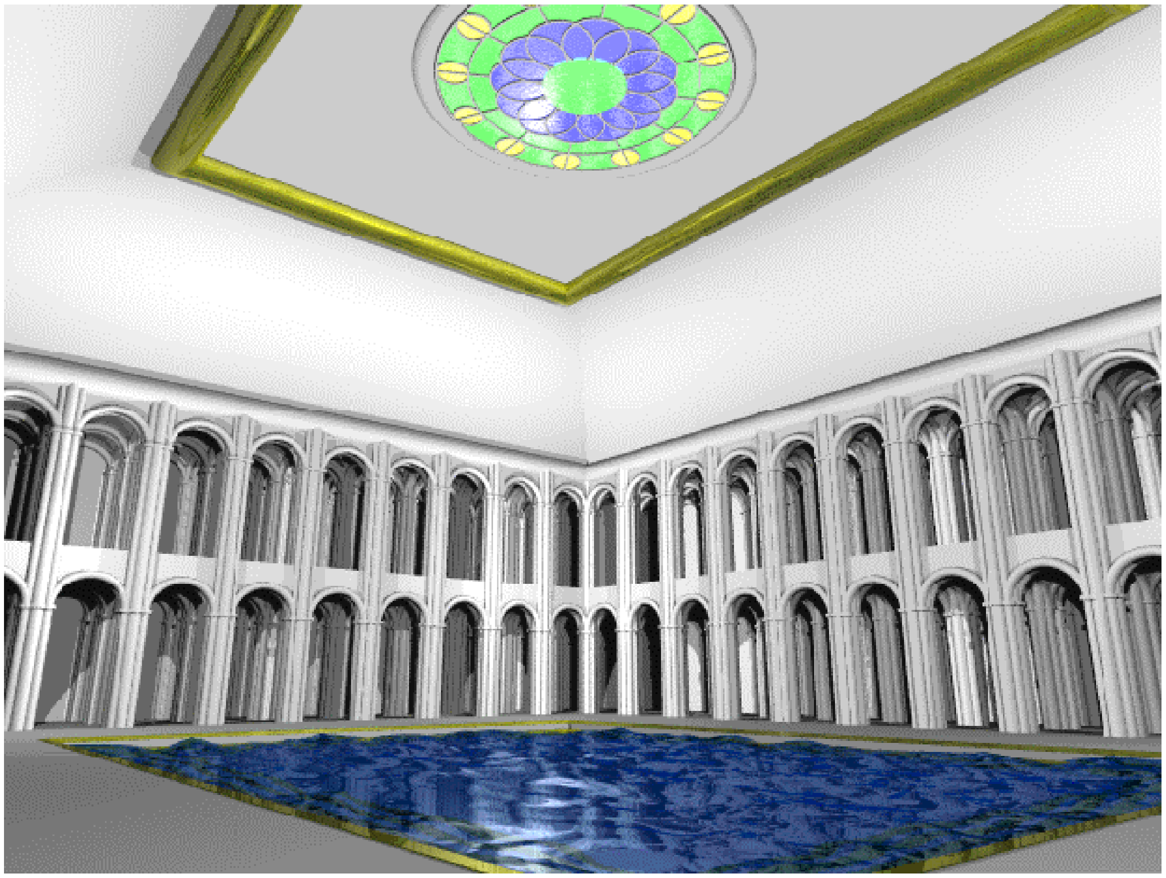
Animation

Once the lighting studies are complete and images have been rendered, an animation was to be generated highlighting what we had found. The intention was to do either a fly-through or an animated change of lighting or a combination of both.

Achievements

Initial Design

When we sat down to actually design the model, we intended to design a room with gothic architecture. The gothic architecture was selected for its innate geometric beauty while maintaining very interesting opportunities for lighting and shadows. However, this idea was soon replaced with a decision to use a Mediterranean architecture instead. The rationale was that Mediterranean architecture was more conducive to a light study. To make gothic architecture realistic, it often calls for dark rooms and sparse lighting. We were more interested in using alternative lighting, including brightly lighting a room -- something that would seem out of place for a gothic piece of architecture. In addition, the Mediterranean style was easier to model with a minimum sacrifice to interesting design.



By the end of the project, we found that we used a bit of gothic architecture irregardless, particularly in the influence over the arches and the vaulted ceiling. However, the room still maintains a very Mediterranean look. The look was loosely based mostly on Charles's memory of architecture seen first-hand as well as "what seemed appropriate." Realism of modeling is not intended to suggest that it the model is based on an actual design.

Selecting a Raytracer

Before we could actually start modeling, we decided that it was best to select a raytracer that was not only capable of doing everything that we wanted, but one which was easy to use. Some research indicated that POV-Ray was the best choice for us due to a number of reasons.

Primarily, it had a rich feature set that was easy to learn yet powerful enough to manifest our design parameters. Its modeling commands were straightforward and easy to use, enabling us to generate a wide variety of complex shapes without having to rely upon specialized modeling tools. In addition, the kinds of lighting and effects that were already built into the program were also well-developed. In particular, we liked its ability to handle media, which is designed to simulate halos, dust, and other atmospheric and volumetric effects. Its ability to handle radiosity as well as raytracing was attractive as well.

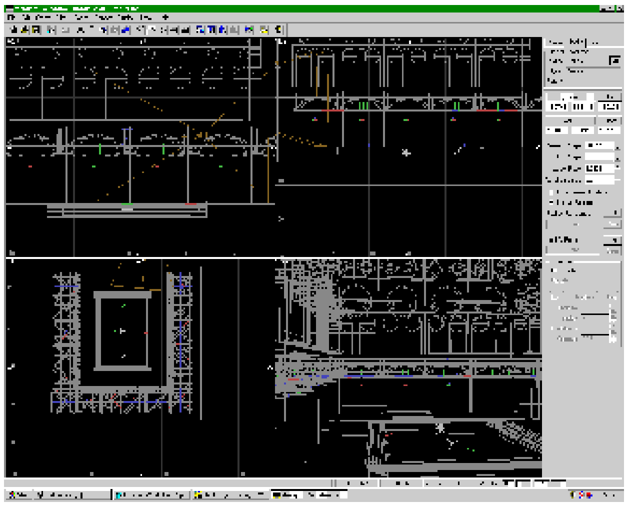
Secondly, POV-ray was freeware. This is an important design consideration, especially for us, because we do not have the resources to purchase expensive commercial packages which can run into the several thousands of dollars. We did not have any other resources available to us, such as existing licensed copies of commercial tools on a workstation that we could use for the duration of the project. Facing these limitations, we were forced to consider only packages that were available to us for low cost. POV-ray, being freeware, was a natural, if not obvious, choice.

Finally, POV-ray was developed so that it could be ported to multiple platforms with ease. With powerful Windows NT and Linux machines available to us in addition to Athena resources, this enabled us to move with ease from machine to machine. The Windows distribution, in particular, was very well designed with a user-friendly front-end and an integrated help system.

We found, to our dismay, that POV-ray's ports were not as useful as we had hoped. We performed 95% of our rendering on a Windows NT machine. The only version available to us for SunOS was version 3.0, as compared to the 3.1 which was available for Windows and Linux. While all the same features were available in both versions, the syntax was slightly different for critical pieces such as textures and media, requiring large amounts of re-coding in order to make them work with eachother. Media in particular was a problem because 3.1 was not backwards-compatible with 3.0. In addition, the Ultra 10s were disappointingly slow. A Windows 98 notebook with a Celeron 400 chip ran almost twice as fast as the Ultras. This quickly shifted us away from using multiple platforms and staying with Windows.

Moray-Modeling Tool and Limitations

Initially, we sought a publicly available modeling package that would work with the POV-ray raytracing software. We discovered "Moray", a commercial modeling package created by a German software company named SoftTronics. A shareware version of Moray was available for a 30-day trial use, which we downloaded and installed. Moray provides a graphical interface for creating, placing, and scaling geometric objects. Our first modeling stage was done using Moray, allowing Moray to deal with the specific mathematics while we focussed on the actual layout and design of the architecture. The following is a screen shot from our beginning work in Moray:



Sample Moray Screenshot

Notice how Moray allows for four different views: the three different planes (x-y, y-z, x-z) and the camera angle. This quality of the software was another benefit; the ability to readily switch between various camera angles and distances was taken advantage of to model our initial architecture.

Once our fundamental architecture, viz., the size and shape of the room, the design of the columns, arches, pool, the floor, was decided, we began to notice some of the limitations of Moray. Since at this point the basic design model of the room was narrowed down, the design advantages of Moray became less important to our progress. When we were faced with placing different pieces of architecture in specific locations with respect to each other, or placing copies of the basic structures at specific locations throughout the room, we wanted to deal with the mathematics ourselves. Especially in the situation where two separate pieces were to touch to exactly form a larger object, the mathematics would have provided us with a more precise method of placement than elementary "eye-balling". Aside from the computational disadvantages of Moray, there was another set of problems. During the phase where we were situating the different structural components, aesthetic appeal was one of the criteria of where objects should be located. This process involved just trying various specific locations and camera angles until a satisfactory model was found. Moray made this process difficult-changing the locations and camera angles by specific quantities was difficult to accomplish by hand.

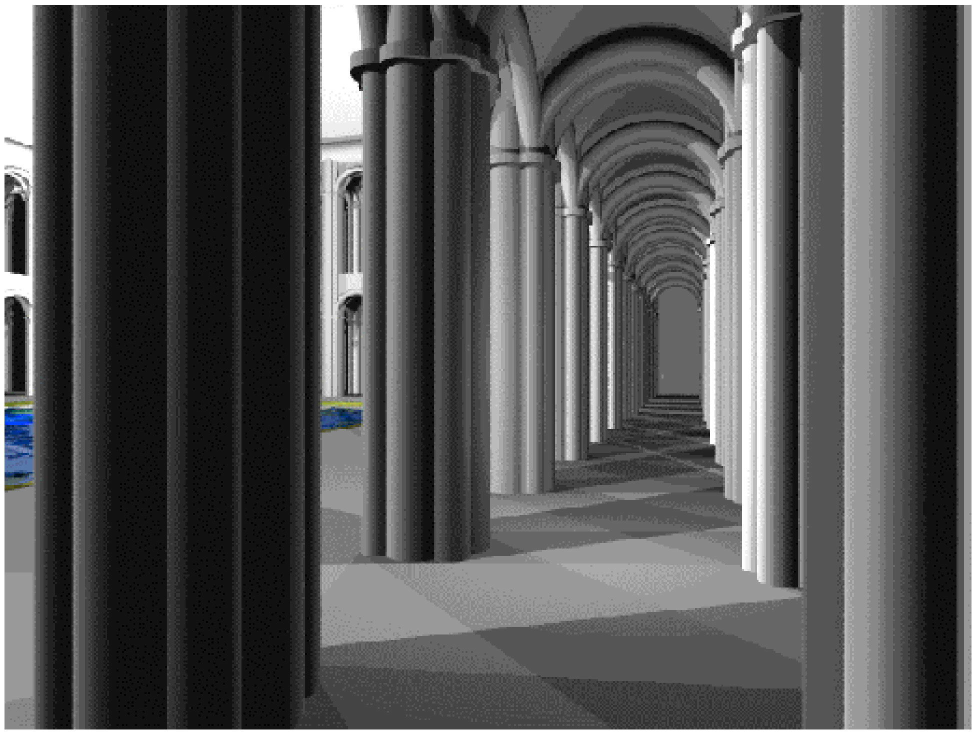
We would, however, soon be forced to stop using Moray by a completely different reason. Though the creators of Moray stated that our version was a 30-day trial free version, what they neglected to mention was that the longer you used a Moray file, the more difficult it would become to render that file. Specifically, Moray would insert a slight pause before rendering that would grow in length proportional to the number of times that the file in use was saved. Although we were starting to move away from using Moray, we were unwilling to lose what we considered a very strong tool in our project development. Buying the software was an option that we were disinclined to take due to its approximate $80 price tag. We spent a few days and considerable resources to try to hack into the Moray file and determine which bits were recording the number of saves, but our efforts did not come to fruition. Reluctantly, we gave up on modeling our file with Moray and started to code directly in the POV-ray file.

Modeling in POV-Ray

Once we decided to abandon Moray and shift to design a model entirely in POV-ray, we thought we could simply export our existing work into a .POV file and work from there. While this did generate the image we were looking for, the sourcefile was completely unmanageable. Every column, arch, and step was its own object -- it did not use referenced objects. As a result, each object was separately defined, making the file huge. In addition, the formatting was readable, but not easy to use. After attempting to fix the file with a bit of editing, it was decided that it would be faster and easier to rebuild the entire scene from scratch.

We found the modeling abilities of POV-ray to be very easy to use, now that we were working directly with the source. Each of the objects was carefully designed in its own space and placed in the world space of the scene. The use of the text file enabled us to make quick modifications to the scene and have its effects propagate through all instances of the object. It allowed for very clear design structure and an object hierarchy. The placement of the objects was made very easy by picking a convenient coordinate system and grid. Selecting a an appropriate scale made it much easier to visualize how the size of certain objects would affect the overall scene without even rendering it first. This was a tremendous benefit to us and made modeling much easier.

Equally as useful was an effective object hierarchy. Our initial model was very crudely made, using mostly slabs to mark columns, arches, and surfaces. This quick sketch gave us a much more precise idea of what we wanted without getting caught up in the details. Once most of our lighting studies were complete, we increased the complexity of the model, adding various elements to the columns, arches, and other surfaces to make a much more detailed picture.



We used many complex CSG (composite solid geometry) operations to create the models that we were seeking, using a combination of unions, differences, and intersections. The entire model was made strictly from CSG operations upon primitives, such as spheres, cyliders, and cubes. The stained glass window, for example, was a complex shape that took advantage of several CSG operations. We were able to design and implement the solution fairly quickly, but found that rendering it took a considerable amount of time. In almost all areas where CSG operations are involved, particularly differences, CSG slowed down the rendering speed.

Because it was very slow at some points, we decided to try to optimize the CSG operations so that the target object would be constructed of fewer CSG operations. In addition, it required some compromises to design. The skylight is a good example of this. The original design was somewhat more complex and used additional cylinders in different locations. However, once we found that this was too slow and optimization meant throwing out parts of the skylight, we did exactly that. The result was a skylight that rendered more than twice as fast at the small expense of complexity of design. Due to time constraints of the project, we were more interested in immediate results than complex modeling.

The columns were originally solid boxes. However, cylinders were added to make the model much more visually appealing. The arches were also simply done: a box with half a cylinder subtracted from it. Portions of toruses were added to give the arches some substance and to help it adopt a more architecturally pleasant appearance. Between each of the columns, the ceiling above it as actually vaulted as opposed to a flat ceiling. Certainly, CSG operations are primarily responsible for the model.

Our model is highly symmetrical and uses a number of repeated segments. Because of the repetition that is used throughout, the precision of the model's objects is absolutely critical to achieving a successful model. Every one of the objects that we used was carefully designed before it was coded to ensure that all the dimensions were correct. When completed, we carefully inspected it to ensure its fit. We are sure that our objects are precisely correct when objects were exactly where they were mentally calculated to be.

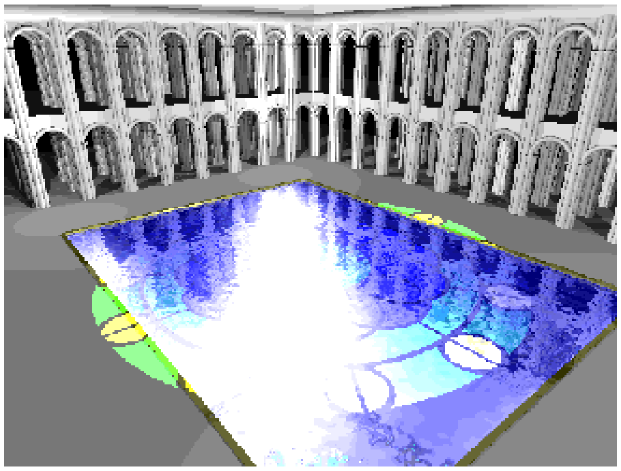
Despite all the precision that POV-Ray gave us through direct manipulation of the text file, we found that we did miss Moray a great deal when it came to move the camera around. To be able to spin a wireframe model around on an axis and then quickly execute a scan-line rendering on the screen to check its finish was tremendous time saver that we did not have in POV-Ray.

We also found ourselves wishing for more powerful modeling capabilities in POV-Ray itself, as well. Primarily, if POV-Ray had a powerful scripting engine instead of a rudimentary object parser, the model could have been completed more quickly using less lines of code. For example, it would have been wonderful to write a script to handle the generation of the skylight or the height field within POV-Ray.

Water

A reflecting pool was a centerpiece of our model that allowed a study of the interaction of light sources and water. Situating light sources at various locations around and inside the pool introduced scenes that dealt with both the reflective and refractive properties of our water model. Our first attempt at water involved creating a box whose properties had two similar material properties of water: color and translucency. A suitable color was determined through trial and error. POV-ray allows for the specification of an index of refraction for a texture, which allowed us to obtain a suitable translucency.

Next we focused on manipulating the surface of this box to create a small body of water with some waves or other similar form of surface activity. These first attempts involved changing the surface normal of the water texture to create models with the appearance of bumpy, wavy, and rippled surfaces. Tested against the columns, this method initially seemed to work perfectly; the reflection of the columns on the water suggested that there was surface activity on our water model.



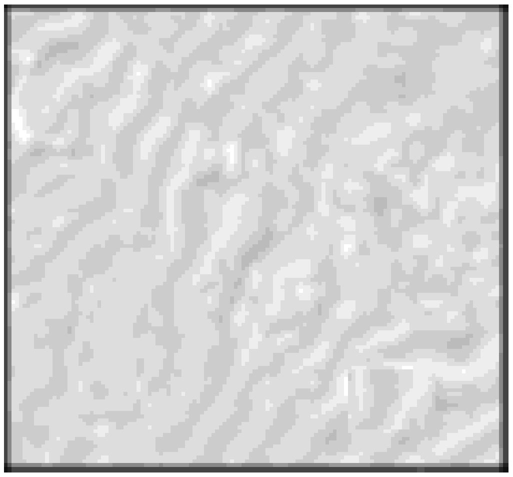
Our original water model (and its key flaw): changing the surface normals.
Although the column reflections look correct, the incident

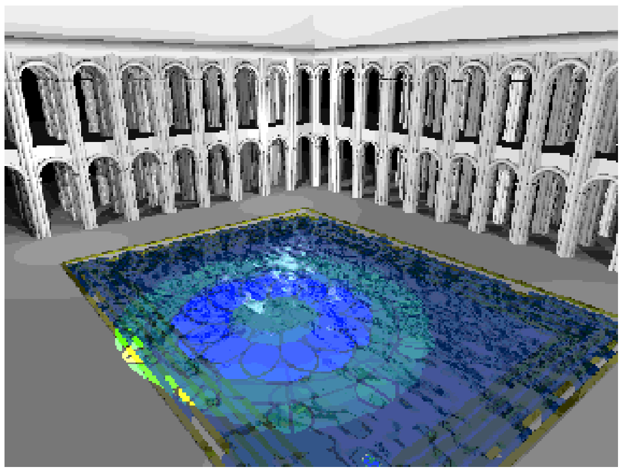
light on the water is undistorted.

To our chagrin, however, testing our water model against the colored light from our stained glass window exposed a major fault in our texture. The light shining through the stained glass window, incident on our water model, should have been distorted similar to the column reflections. What we discovered, however, was that manipulating the surface normals created the appearance of surface activity by distorting reflections. Since the incident colored light was not a reflection on the water, the image of the stained glass window on the surface of our water model was undistorted, discrediting our water model.

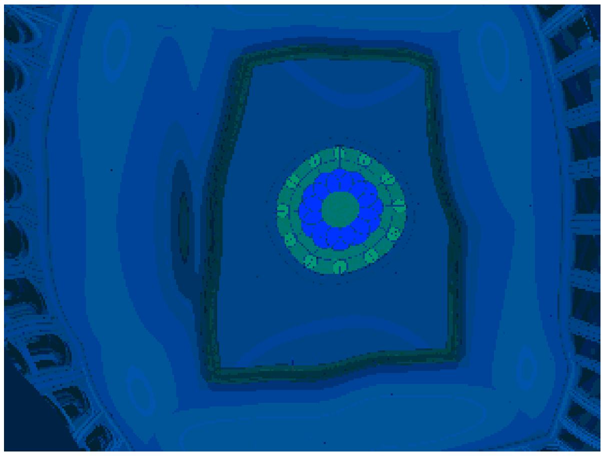
Realizing that we would need to specifically change the height of water object, we started to implement a height-field. We looked at existing water models on the Internet to determine the best way to implement a height field. A common implementation was to render an image of a water object with a manipulated surface normal. Attempting to create our own height field image, we rendered our water model with the camera directly above the water, line of sight perpendicular to the water surface. Loading the resulting image in a graphics creation package (Adobe Fireworks) we smoothened out the image to avoid large differences in water height. Despite our "touching up" efforts, the resulting water looked too turbulent and differences in water height remained large. Scaling the height-field accentuated small height differences in our height field.

Running out of time, we decided to search for either a height field generator for water models, or an existing height field image. Most of the height field generators that we examined were designed to create height fields for various mountainous terrain-clearly unsuitable for a water model. Eventually we found a height field image in an existing water model. We touched up that height field image and placed it as our water model, obtaining satisfactory results.


The Height-Field Image


Resulting water with Incident Stained Glass Light

We were not sure how the height field affected the translucency of the water model. As the height field already properly distorted light incident to it, we sought to further check whether the incident light was properly refracted. Placing a camera inside the water model, looking upwards at the stained glass window, we rendered the following image. It reveals the smoothness of the surface of our water model. That is, despite the fact that the height field generated is a collection of discrete heights, the resulting height field appears smooth. This smoothing algorithm is built into POV-ray and operates by extrapolating the heights in between the discrete points.



Camera inside Water

Stained Glass

Another centerpiece of our model is the circular stained glass window directly above the pool. Its creation served as an avenue for the study of the interaction of white light and colored glass, colored light and water, and colored light and the stone texture. The first step in creating a stained glass window was designing the frame. By subtracting a smaller cylinder from a larger one, a ring was created. Two rings were then connecting by a thin narrow bar. This base image then was copied and rotated around 12 times, forming the full circular frame. Designing the frame was done entirely in POV-ray, because by this point of time we had abandoned Moray. Coding directly in POV-ray proved beneficial, however, due to the reasons mentioned earlier. After the initial ring was made, generating the base image and then the frame was an execution of precise placement and multiple rotation.

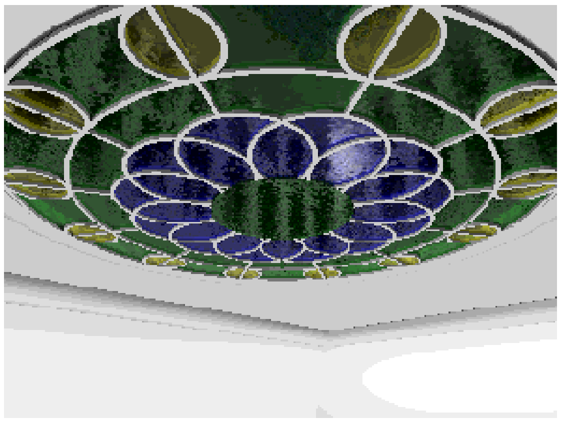
Designing the glass that was to be in this frame was a more difficult task. As a stained glass window, our window required multiple colors of glass. We also desired to keep the center hole of the whole window without glass, to permit the unobstructed passage of light. The inner set of rings was given one glass color, while the outer rings were given another. Then, both sets of rings, as well as the inner circle were subtracted from a larger cylinder the same size of the window. This subtraction allowed for the specification of the texture (and color) of the remaining piece of the window. To give the appearance of true stained glass, we eliminated the image of the perfect smoothness of the glass by creating a bumpy surface normal.

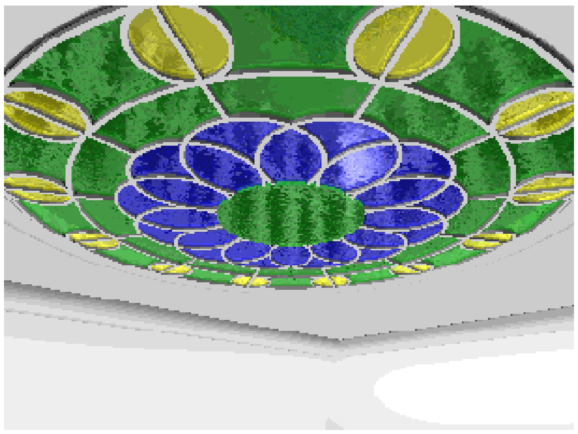
The stained glass window was placed in a flat ceiling 40 units above the pool. POV-ray does not permit the creation of infinite light sources-due to round-off errors, light sources in POV-ray are unable to calculate distances from very distant objects. To create the impression of an infinite light source, a spotlight light source was placed 160,000 units above the window, which was within the problematic distance threshold. Though we were unable to obtain perfectly parallel rays of light, the resulting rays from this distant light source appear parallel to the human eye.

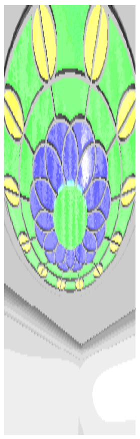
Light passing through this stained glass window caused colored light to be incident with the various objects in the model. When colored light was incident on the pool, we were able to gauge the accuracy of our pool model-and this ability revealed to us when our initial water model was incorrect (as discussed above). When the light source was placed not directly above the window, but offset a little to one side, the resulting colored light was incident on both water and the stone. This image reveals the different natures of these two materials.

Despite all this correct behavior of the interaction of the colored light and the objects below the window, one key aspect of the stained glass window was missing: the illumination of the stained glass itself. Though the stained glass was properly allowing light to pass through, the light itself looked rather dull. We increased the specular-reflection coefficient of the glass material, under the belief that the increased specular reflection on one side of the glass would illuminate it. When this act did not fix the problem, we entered a parameter in the material that introduced the Phong specular-reflection model to the glass. Again, however, tweaking of this parameter did not produce the desired results.

Befuddled and under a time constraint, we hacked the stained glass model by increasing the amount of ambient light reflected to match the intensity of the light source behind the stained glass window. We are still unable to explain why our attempts to properly illuminate the stained glass did not succeed. Given more time, however, we could have analyzed this problem more thoroughly, instead of looking for a quick solution. The resulting stained glass window, albeit hacked, does however look convincing.


Stained Glass Window without Ambient = 0

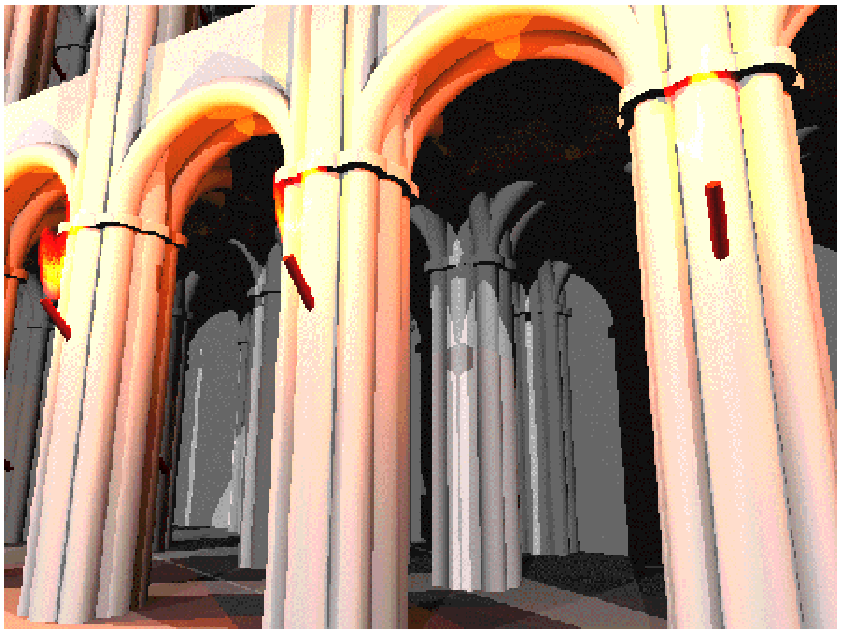

Stained Glass Window without Ambient = 1


Stained

Media -- POV-Ray Atmospheric and Volumetric Effects

In POV-Ray version 3.0, two new features were added: halo and atmosphere. The first dealt with effects within a solid object, such as water, while the second was designed to simulate particles in the air, such as water vapor or dust. In version 3.1, these two were generalized to become media. Media could be applied to any object as well as the scene in general, allowing for objects that were semi-transparent throughout the volume of the object, not just at the surface. Lights that do not normally diminish over distance could thus be attenuated by media. We used media for both the scene and objects.

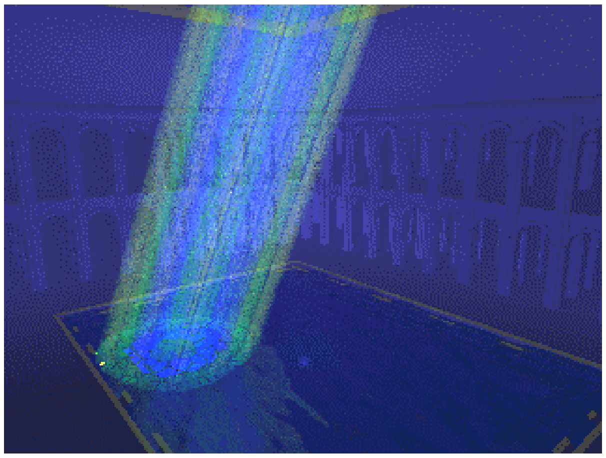
We used media to create the flames in the torches. Because media defined the shape of the flame inside more than providing light, a point light source was placed inside the flame to actually produce most of the lighting. The flame itself was a largely teardrop-shaped object which was a translucent red. The media was used on a separate object inside the flame to create the yellow core of the flame. Its ambient light was turned to a magnitude of 5 to make it opaque and to give it the appearance of a blinding source of light.



The result is an acceptable torch flame, but certainly not amazing. Using hindsight, a great torch flame would have been using a density map upon an object with a combination of media. This would have made the flame mostly shapeless. Realistic flames are a challenge to many ray tracers, we found, and messages on Deja.com and bulletin boards seemed to reflect this. However, given the period of time that we had, we felt that we made the most effective compromise.

The scene’s media was designed to simulate dust in the air. We chose to render the skylight’s light with media. The use of multiple colors in the skylight as well as its position in the room allowed us to create a very interesting scene with a visible column of light. The use of different colors is entirely visible and the shape of each portion of the stained glass can be discerned not only in the column of light, but also on the surface below, both on the water and off. While this media is rather dense and somewhat coarse, it clearly illustrates the capabilities of POV-Ray's atmospheric media.

Experiments with non-directional light sources proved to be less impressive, as it tended to appear more like fog than anything else. The stark contrast created by directional lights and the side-by-side difference between the lighted and unlighted media allows for a great deal of artistic freedom, which we were not able to fully explore. We wanted to create several more lights, but time constraints prevented this.



Media is extremely slow to render. The skylight image with the column of light took nearly 45 minutes to complete at only 320x240 without any anti-aliasing. In addition, the quality of the media was turned down to complete the images in a reasonable amount of time. Turning the quality of the media up (which improves the granularity of the rendered media and creates a generally smoother look), caused the speed to drop to less than 2 pixels per second. Compounded by the complex CSG functions in the room, this was impractical.

Area Lights

Area lights in POV-Ray are simply an array of existing lights whose shadows are averaged to achieve soft shadowing effects. While they achieved believable and notable results, the time required for rendering was beyond what we could afford in our time constraints. As a result, after some experimentation, we decided that we were more concerned about accurate lighting with other more conventional lights.

Radiosity

Radiosity would have been a very welcome addition to our collection of images. However, due to the nature of the project, with many scenes containing dozens of lights, we thought that radiosity would have been temporally prohibitive. As a result, no images were tested or generated with radiosity. We would, however, in the future like to take our model and run a radiosity trace on it to see what it looks like. We do have a concern that due to the number of lights in the model, that it could prove to be too bright, especially with the light colored texture. However, this will require further experimentation.

Animations

Our proposal called for the generation of an animation to show our work. While this would have been incredibly interesting and very appealing to view, we decided to not pursue this within the time constraints. We originally underestimated the quality and importance of still rendered images. By focusing only on stills, we were able to explore lighting scenarios in many ways that we believe would have been overlooked had we pushed to create an animation within the constraints.

Individual Contributions

- Chaz Yoon:
    - Development of details of Architecture (trim on columns, vaults, ceiling)
    - Stained Glass Window Model (frame, glass, lighting, media)
    - Flaming Torches Model
- Ajay Kulkarni:
    - Development of fundamental Architecture (arches, pool, layout)
    - Standing Water Pool Model (both attempts, height field use)
    - Various lighting scenarios

Lessons Learned

Modeling

The fundamental obstacle we encountered was trying to realistically model different objects. In class, we dealt with basic issues in modeling objects, and then we moved onto the algorithmic backbone of graphics rendering-the issues involved in designing for realism were never really examined. Due to this background, initially we were under the belief that some of these objects would be relatively straight forward to create using the proper geometry and texture. One good example of this initial fallacy behind our modeling is our water object. We first just created a box, gave a color, set the index of refraction, and changed the surface normals to resemble water. Early inspection of our water model with reflection seemed to suggest that our model was sufficient. With the addition of the stained glass window and lighting, however, our eyes were soon opened to the various problems involved.

In our endeavors to realistically model the primary components of our design, we specifically dealt with the following key issues:

- Crucial advantage of height fields over surface normal manipulation: the ability to generate smooth, continuous objects of changing height
- The use of media to produce atmospheric and volumetric effects (the dust and torch flames, respectively)
- Effective use and optimization of Composite Solid Geometry (CSG) (in the arches, vaults, columns, etc.)
- Differences between the translucent and reflective natures of objects (water and stained glass)

The wide user base of POV-ray proved to be an important aide in our modeling endeavors. Although we had not placed much emphasis on this characteristic while choosing renderers, in retrospect this aspect of POV-ray gave us a few "kick-starts" when we were experiencing difficulties in our modeling attempts at realism.

Lighting

Our light study gave us a newfound respect for optics. We recognized that realistic lighting involved more than just situating a light source at the right location. Other factors involved in lighting models involve the intensity and color of the light, as well as the different ways to introduce lighting into a model. The stained glass window and the torches are examples of more complex ways to illuminate a room. We then realized that in order to create a realistic model, more than realistic objects is required-a realistic lighting scenario must also be created. For example, we created our stained glass window and shone light through it. At first, we were satisfied that light was properly being filtered through the colored glass. However, without the illumination of the stained glass window, the stained glass model was not believable.

Project Management

One of the biggest problems that teams who work together face is dealing with proper division of labor, communication, and making the final piecing together relatively seamless and straightforward. Even with only two people in the group, our team was no different. Initial problems involved deciding on a common scale and geometry of the architecture. To avoid inconsistencies and overlap in our work, we were able to stagger working on the project so that only one person was working on the project at a given time (while the other person dealt with his other classes). After the primary architecture of the room was designed, however, the work that remained lent itself to modular development. At this stage of the modeling, the more important (and more difficult) objects were to be designed. We assigned specific objects to each member of the group. Periodically, however, we would touch base and compare and contrast problems and other issues. We found it helpful to look at the other member?s objects and provide feedback and suggestions for improvement.

Acknowledgements

- TA Kari Anne Høier Kjølaas - for providing those weekly jumpstarts
- *Jaime Vives Piqueres - for the height field image and examples*
- Nathan O? Brien - design inspiration
- Marcus Dos Santos - design critique
- Leroy Walker - design critique

Bibliography

- The Internet Raytracing Competition (http://www.irtc.org)
- POV-ray Homepage (http://www.povray.org)

- Moray Homepage (http://www.stmuc.com/moray/)
- Usenet/Deja.com Discussion Forums (comp.graphics.raytracing)
- Course Textbook: Hearne, Donald; Baker, M. Pauline; "Computer Graphics C Version"; Prentice Hall, 1996, pages 494-511

Appendix

- POV-ray Source code