Synthetically Real Graphics

Khalid El-Arini
Advised by Todd Mowry

April 30, 2004

Abstract

Synthetic Reality is a project with the goal of creating metamorphic robotic systems that simulate real moving objects. These systems consist of small, independent modules, or "catoms," each containing a processor, memory, a capacitor (for power), programmable magnets (for motion), and both wireless and wired communication capability. The surface of each catom is equipped with a graphical display, on which appropriate textures can be mapped to simulate the surface of the real object that is being modeled. This project addresses the problem of mapping textures to the surfaces of catoms so as to minimize distortion from the original model. Depending on the catom size, shapes to be modeled by a system of catoms will potentially lose much of their topographic information. As a result, texturing is very important in maintaining an approximation of the original characteristics of the model.

1 Introduction

1.1 Synthetic Reality

Synthetic Reality is a project recently started by Carnegie Mellon professors Seth Goldstein and Todd Mowry with the goal of creating metamorphic robotic systems that simulate real moving objects. These systems consist of small, independent modules, or "catoms," each containing a processor, memory, a capacitor (for power), programmable magnets (for motion), and both wireless and wired communication capability. The surface of each catom is equipped with a graphical display, on which appropriate textures can be mapped to simulate the surface of the real object that is being modeled.

Catoms will be spherical in shape, but their eventual size is heavily dependent on the locomotion and communication mechanisms needed for them to interact with each other. Although initial three dimensional prototypes are expected to be the size of ping-pong balls, advancements in nanotechnology over the next several years could reduce their size significantly. The Synthetic Reality research group is currently working on two dimensional prototypes that operate on a power grid placed on a flat surface.

Applications of this project range from the field of medical robotics to the sports entertainment industry, and beyond. For instance, combining a Synthetic Reality system with live image capture and actuators allows a surgeon to physically manipulate a scaled up, catom version of the blood vessels and tissue in question, and have the deformations replicated in the patient's body. In another example, a live, three-dimensional video feed of a football game can be scaled down to a tabletop system of catoms, allowing the sports fan to watch the game in three dimensions without actually being present at the stadium.

1.2 Problem Definition

This paper addresses the problem of mapping textures to the surfaces of catoms so as to minimize distortion from the original model. In this context, distortion is defined as the per pixel difference between a view of the original object and a view, from the same viewpoint, of the object rendered with catoms. Depending on the catom size, shapes to be modeled by a system of catoms will potentially lose much of their topographic information. This occurs if the diameter of the catoms is larger than the smallest feature size of the original model, thus providing resolution too coarse to produce a faithful replica. As a result, texturing is very important in maintaining an approximation of the original characteristics of the model.

Since it will be some time before three dimensional physical prototypes are built that include a graphical display, this research was done using a simulation environment implemented by the author. In this environment, catoms are colored and shaped based on a passed in object model, consisting of a mesh and one or more textures. Experiments were done where each catom was restricted to only one color, and where each catom was textured with multiple colors. Furthermore, the author investigated both the case where the location of the viewer is known, and where the location is unknown, resulting in view-dependent and view-independent algorithms.

1.3 Simulation Details

The Synthetic Reality Graphics Simulator is an OpenGL application that renders object models passed in OBJ format as catoms, textured with the passed in image. This is different than directly texturing the catom-rendered object with an image, since in order to mimic the appearance of the original object, appropriate texture coordinates must be extracted from the object model. Throughout this work, results were obtained by running the simulator on an elephant mesh, with two textures: a checker pattern and a blue wavy pattern. Runs were performed with catoms of various sizes, and were compared to baseline images generated in Maya of the original OBJ model, as shown in Figures 1 and 2.

The simulator involves a preprocessing step of obtaining catom locations from the original mesh, given a catom radius. Each catom is stored as a vector representing the center vertex, and the associated triangles in the original mesh that are contained within this catom. Similar to the marching cubes algorithm,

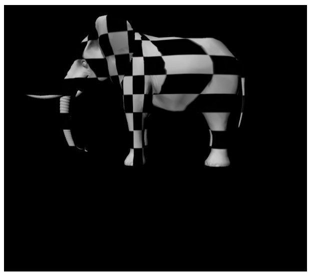
Figure 1: Baseline Maya Image (checker texture)

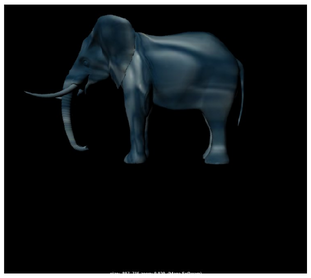
Figure 2: Baseline Maya Image (blue texture)

this process works in voxel-space, where the voxels are the size of catoms. If a triangle from the original mesh intersects a specific voxel, a catom is placed in that location.

1.4 Related Work

Traditional graphics literature does not specifically cover the problem investigated in this work, but there are a few areas in computer graphics that attempt to solve similar problems. Prime among these is point-based rendering, or using sample points from surface models as display primitives. Although the motivation for point-based rendering is efficiency when dealing with large, complex models, this work can be seen as a three dimensional variant of that idea. In [5], the authors use a point-based rendering system in a large-scale three dimensional digitization project to render models containing hundreds of millions of samples. Others extend the technique to transparent objects as well, using Elliptical Weighted Average (EWA) texture filtering [7]. In [6], the authors create an editor for three dimensional point-sampled geometry, in order to explore the usability of point primitives for surface editing.

Parameterization is another area of computer graphics that is relevant to the problem investigated in this paper. Parameterizing a three dimensional mesh involves computing a correspondence between a surface patch on an object and a planar mesh, such as a texture. The authors of [1] investigate parameteriza-

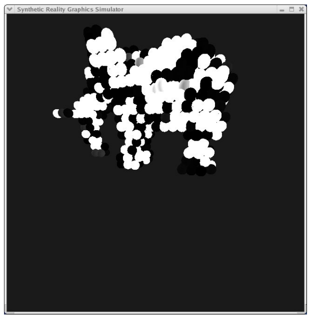

Figure 3: Initial Single Color Implementation (checker texture, large catoms)

tions of textures that minimize distortion of different intrinsic measures of the mesh. In [4], the authors propose a technique for parameterizing textures onto a spherical domain, while minimizing a stretch-based measure in order to reduce distortion.

2 Single Color Catoms

The earliest work on this problem revolved around the idea of coloring each catom with a single color, in order to avoid the problem of assigning appropriate texture coordinates to the catom model. The first attempt at single color catom coloring consisted of coloring the catom with the same color as a randomly chosen triangle from within that catom. More precisely, each spherical catom can be associated with a set of triangles in the original mesh that would intersect the sphere if both models were superimposed on one another. For each catom, one of these triangles is randomly chosen, and then the colors at each of the triangle's vertex are averaged in order to obtain the color of the catom.

Results from this method are shown for the elephant model (Figures 3 through 6), textured with both the checker pattern and the blue pattern, and rendered using two different catom sizes. Note that the checker pattern is barely discernible.

The next attempt at a single color catom technique was a stratified sampling method, which divides the surface of each catom into 32 patches, based on spherical coordinates. Namely, ϕ ranges from $[0, \pi]$, and θ ranges from $[0, 2\pi]$, both with a step size of $\frac{\pi}{4}$. Next, a point $\mathbf{p}$ is randomly (uniformly) chosen from within each patch, and a ray is shot out from $\mathbf{p}$ to the corresponding location of the center of the catom on the original mesh (Figure 7). (The reason for the randomness is to avoid aliasing due to regular patterns, such as checkers or stripes.) If shooting the ray results in a front-side intersection [2] with a triangle in the original mesh, the appropriate color of the intersection point is interpolated (using barycentric coordinates), and is averaged uniformly along with the colors associated with all other front-side intersections for this catom, in order to generate the catom's final color.

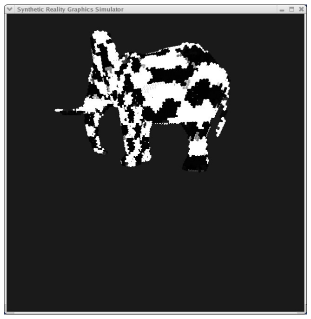

Figure 4: Initial Single Color Implementation (checker texture, small catoms)

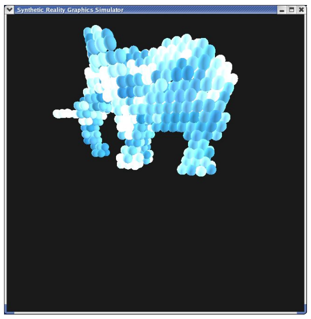

Figure 5: Initial Single Color Implementation (blue texture, large catoms)

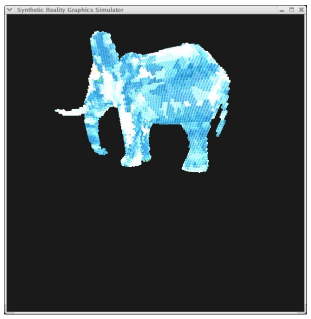

Figure 6: Initial Single Color Implementation (blue texture, small catoms)

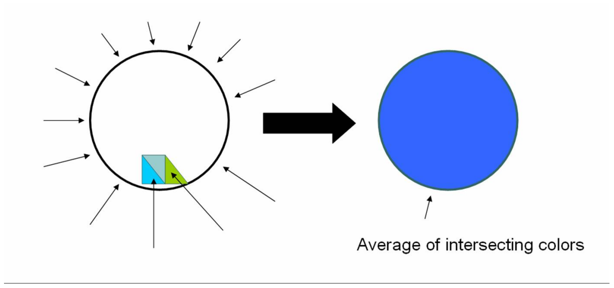

Figure 7: Stratified Sampling Single Color Implementation (methodology)

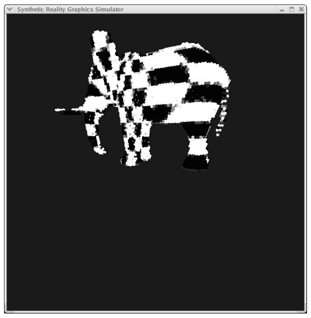

Figure 8: Ray Traced Single Color Implementation (checker texture)

These results, shown in Figures 8 and 9, are much better than the previous ones. It is also the case that dividing the catom surface into more than 32 patches does not significantly increase accuracy, but increases computation time.

Another final attempt at single color catom coloring was made, but this time taking into account the position of the viewer. This algorithm proceeds exactly as the previous ray tracing algorithm, except that instead of uniformly averaging the color contributions from each intersection point, a color now gets more weight in the average if the dot product between the ray that produced it and the ray representing the view angle is greatest.

These results barely differ from the previous ones, although it is noteworthy that some of the visual artifacts (e.g. improperly colored holes in the middle of the model) have disappeared (Figures 10 and 11).

3 Textured Catoms

More faithful textured models can be created if catoms are allowed to assume multiple colors.

One such method is similar to the ray tracing based approaches from the previous section. Like before, the surface of the catom is subdivided evenly into patches. However, instead of randomly selecting a point within each patch

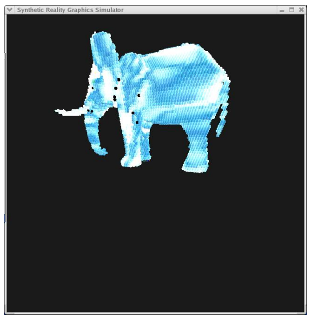

Figure 9: Ray Traced Single Color Implementation (blue texture)

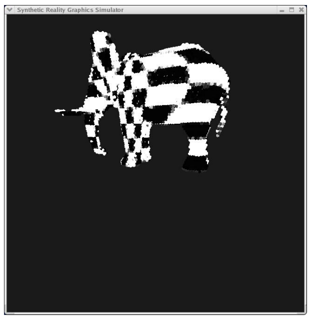

Figure 10: View Dependent Single Color Implementation (checker texture)

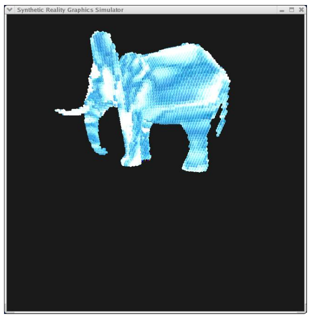

Figure 11: View Dependent Single Color Implementation (blue texture)

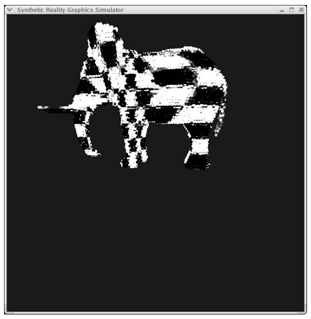

Figure 12: Textured Implementation (checker texture)

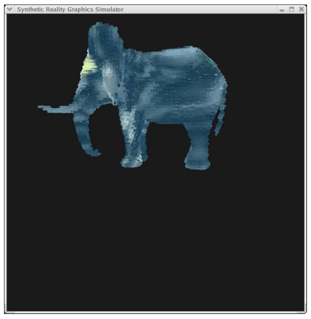

Figure 13: Textured Implementation (blue texture)

to shoot the ray from, rays are shot evenly in a grid from points **P** across the surface of the catom to the center of the catom. Rather than extracting color information from the intersection points, we instead extract texture coordinates from the original model, which we directly map onto the spheres.

The results are shown in Figures 12 and 13. Note that this is the closest that the blue texture on the catom model has looked to the original blue texture on the baseline image.

One can take viewer location into account, and shoot the rays not from the surface of the catom to the center, but rather from the viewpoint through the surface points **P**, as shown in Figure 14.

This technique produces very accurate results, which are displayed in Figures 15 and 16. This iteration of the model with the checkered texture is the first one to display straight lines.

4 Conclusion

It is important to note the performance requirements for this system of catoms. Any algorithm chosen to color the catoms must be distributed and local, in the sense that each catom can compute its color or texture with minimal, if

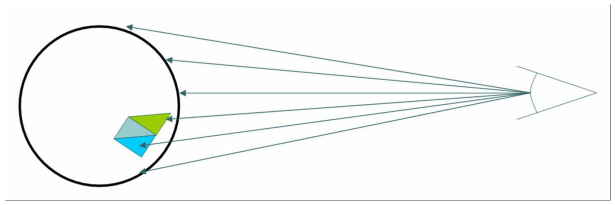
Figure 14: View Dependent Textured Implementation (methodology)

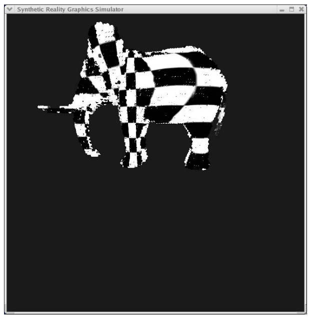

Figure 15: View Dependent Textured Implementation (checker texture)

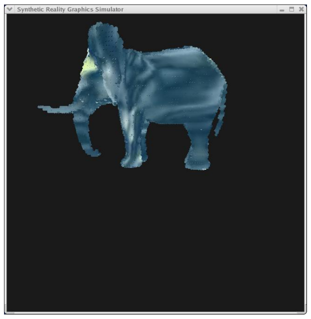

Figure 16: View Dependent Textured Implementation (blue texture)

any, interaction with other catoms. A catom certainly cannot afford to wait for information from all of the other catoms in the system, since this clearly does not scale with the complexity of the model. All of the algorithms explored in this work meet these performance requirements. A catom only needs local texture information in order to shoot rays at the original model and retrieve appropriate colors. Experiments were done in the view dependent case where rays were shot at all of the triangles in the mesh, not just the local ones, and there was only a negligible increase in quality.

In the near future, the multicolored textured algorithms will provide the most realistic rendering of colors on catoms, since the catoms themselves will be relatively large in size. In this scenario, using the single color techniques would produce severely under-sampled textures. As the field of nanotechnology advances, however, and the catoms become smaller, the simple single color catom algorithms will suffice, since each catom will have the resolution of a single pixel.

5 Future Work

It would be interesting to investigate the effects of lighting on the surfaces of the catoms, and to perhaps compensate for external lighting effects by changing the hue or brightness of the catom's color. This situation may arise if a three dimensional video feed is used as input to create the catom model, and the source lighting is vastly different than the lighting in the room where the catoms are located.

Another interesting problem is that of view dependent algorithms that take into account multiple viewers, instead of just one. How do these techniques extend to the two viewer case? It may be the case that after a certain number of viewers, the benefits obtained by using the view dependent algorithms disappear, and the view independent algorithms would be preferable due to ease of computation.

6 Acknowledgements

The author would like to thank Todd Mowry, Seth Goldstein, Ben Rister, James Kuffner, Nancy Pollard and Robert Wang for their contributions to this work.

References

[1] M. Desbrun, M. Meyer, and P. Alliez. Intrinsic parameterizations of surface meshes. In *EUROGRAPHICS*, 2002.

[2] A. S. Glassner. *An Introduction to ray tracing.* Academic, London, 1989.

[3] P. S. Heckbert. Survey of texture mapping. *IEEE Computer Graphics and Applications*, pages 207–212, 1986.

[4] E. Praun and H. Hoppe. Spherical parametrization and remeshing. *ACM Trans. Graph.*, 22(3):340–349, 2003.

[5] S. Rusinkiewicz and M. Levoy. Qsplat: a multiresolution point rendering system for large meshes. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 343–352. ACM Press/Addison-Wesley Publishing Co., 2000.

[6] M. Zwicker, M. Pauly, O. Knoll, and M. Gross. Pointshop 3d: an interactive system for point-based surface editing. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 322–329. ACM Press, 2002.

[7] M. Zwicker, H. Pfister, J. van Baar, and M. Gross. Surface splatting. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 371–378. ACM Press, 2001.