Crowd Rendering with Non-Planar 3D Impostors

Jerry Yee and James Davis
Department of Computer Science
University of California, Santa Cruz

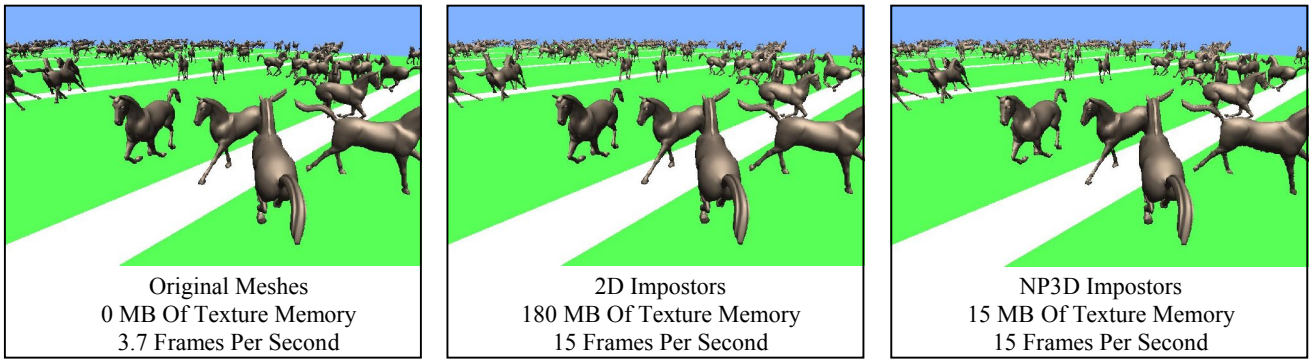

Figure 1. Our NP3D impostor method renders crowds using significantly less texture memory than the 2D impostor method, while maintaining visual quality identical to the original meshes. The impostors shown here were rendered using 64x64 pixel image sizes. Because NP3D impostors require fewer images, its memory consumption is less than a tenth of the 2D impostor's consumption.

Abstract

We present a crowd rendering method that can animate hundreds of characters in real-time. Existing methods utilize level-of-detail (LOD) schemes such as geometric simplification and impostors, which often consume large amounts of system resources and introduce visible animation artifacts. Popping artifacts are especially visible when the camera angle changes. We propose a system that utilizes non-planar 3D (NP3D) impostors in a crowd simulation that can be viewed from any perspective with less computing resources and popping artifacts than existing methods. For characters near the viewer, high-resolution 3D meshes are rendered to provide detailed renderings. Distant characters are rendered using NP3D impostors, which are basically images projected onto low-resolution versions of the original high-resolution meshes. Our system is designed for use in interactive virtual environments, running on the average home computer system.

1. Introduction

Creating realistic crowds is an important feature of many interactive virtual environments. Many games simulate crowds of people that interact with each other like real people. These games must animate and render hundreds of characters in real-time. The camera must move about the scene seamlessly in order to provide an immersive experience. At animation studios, artists and engineers may want to sketch out crowd scenes and change camera angles interactively for storyboarding and planning. Interactive crowd simulations would be an invaluable tool for this purpose. However since a single character may consist of thousands of polygons, significant slowdown occurs when more than a hundred characters are animated on current computer hardware. Simplification methods are needed to reduce the number of rendered polygons in order to animate the scene in real-time.

To animate hundreds of characters, current methods adopt level-of-detail (LOD) schemes. Characters close to the camera are shown without any compression or reduction of image quality. However as characters move farther from the camera, they become smaller and require less details. Several detail reduction methods have been explored, such as geometric simplification and image-based LOD methods. Geometric simplification methods introduce awkward skeletal animations, and they use large amounts of memory to store joint reduction data. Image-based methods must make a tradeoff between introducing animation popping artifacts and consuming lots of texture memory storing their images. Because they consume a lot of resources, they are unusable in large game engines which reserve most of their GPU resources for other purposes.

We present a new non-planar 3D (NP3D) impostor LOD rendering method and demonstrate it in a working system. NP3D impostors are images projected onto low-resolution versions of the original high-resolution meshes. Since the impostor is not a flat image, a single impostor can be viewed from multiple perspectives. This increases

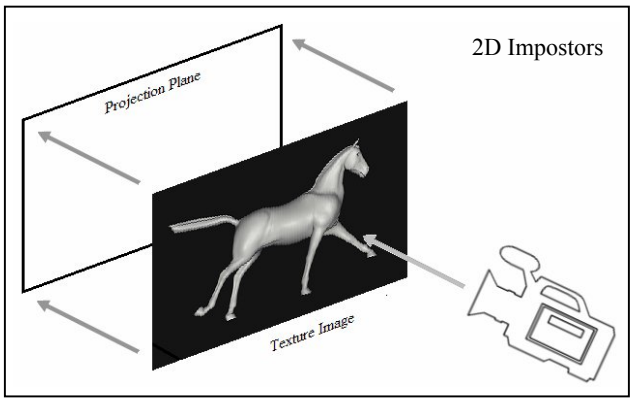

Figure 2. 2D impostors are created by texture mapping an image of the character onto a quadrilateral projection plane that faces the camera.

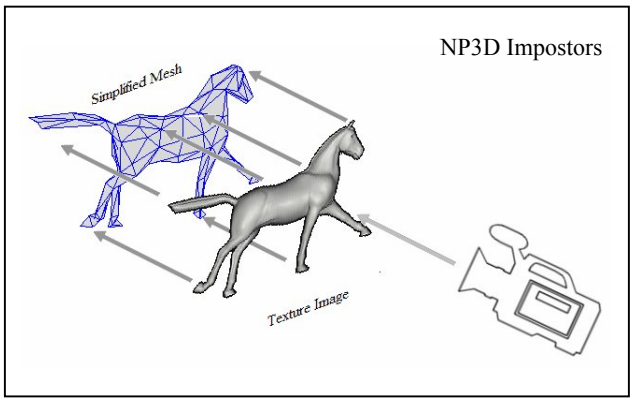

Figure 3. NP3D impostors are created by texture mapping an image of the character onto a simplified version of the original mesh.

the validity and life of a single impostor which offers distinct advantages over traditional 2D impostors.

Our system provides an interactive, real-time simulation. It is capable of animating and rendering hundreds of characters with minimal artifacts and less texture memory than other systems. The camera moves seamlessly about the scene, from directly overhead to a side profile of the scene, and anywhere in between. Characters near the camera are rendered using the original high-resolution 3D meshes. For distant characters, we render them using NP3D impostors.

The main contribution of our work is a new crowd rendering method using NP3D impostors that can be implemented in a real-time crowd animation system. We demonstrate this method in a simple crowd system and analyze the improved texture memory usage and frame rate relative to 2D imposters.

2. Related Works

Geometric simplification is well studied for the purpose of crowd simulation. Ahn [5][6] reduces motion complexity by reducing the number of articulated skeletal joints. However these motion simplifications introduce visible animation errors as characters move closer to the camera. Missing joints and limbs become highly visible. Hoppe [1], O'Sullivan [4] and Wand [8] use mesh simplification to reduce the number of polygons in their key frame meshes. Farther characters are replaced with progressively simpler meshes. However these meshes are expensive to store and regenerate during continuous crowd simulation.

Image-based rendering methods, in which geometries are replaced with 2D impostor images, have also been investigated, e.g. Aubel [2], O'Sullivan [4], and Dobbyn [7]. For every camera viewpoint, impostors are pre-generated thus diverting complex geometric rendering to a preprocessing step. This enables real-time animation during the actual simulation, because impostors are rendered in place of complex meshes. However impostors are view dependent, so the transition from one impostor to another is not smooth and distracting popping artifacts occur. Furthermore the impostor images are stored in GPU texture memory, which have limited capacities and must be shared with other textures. Because impostors are collected for every desired viewpoint, the texture memory space fills up quickly.

Some advanced impostor methods aim at reducing the amount of required texture memory. Tecchia [3] explore texture compression, a more efficient manner to store impostors. However the storage capacity remains bounded, and it is not possible to store impostors for every possible camera viewpoint. Dynamic impostor generation is explored by Schaufler [12], but this is bounded by the available processing power. During continuous crowd simulation, there is a very short time frame in which to generate these impostors. It is usually not possible to dynamically generate the impostors without animation lag. So these systems either reduce the resolution of their impostors or reduce the total number of impostor images. The former increases the distance at which the impostors may replace the 3D geometries. The latter increases the angular separation between each successive impostor, thus increasing the occurrence and severity of popping.

View-dependent texture mapping has been used to create virtual walks through city landscapes. Debevec [9], Sillion [10], and Decoret [11] describe methods that allow for multiple views of buildings. They recreate simple geometric building meshes and then texture map images onto each face of the buildings. This works well for buildings having well-defined shapes and planar faces. However its application to human and animals has not been explored.

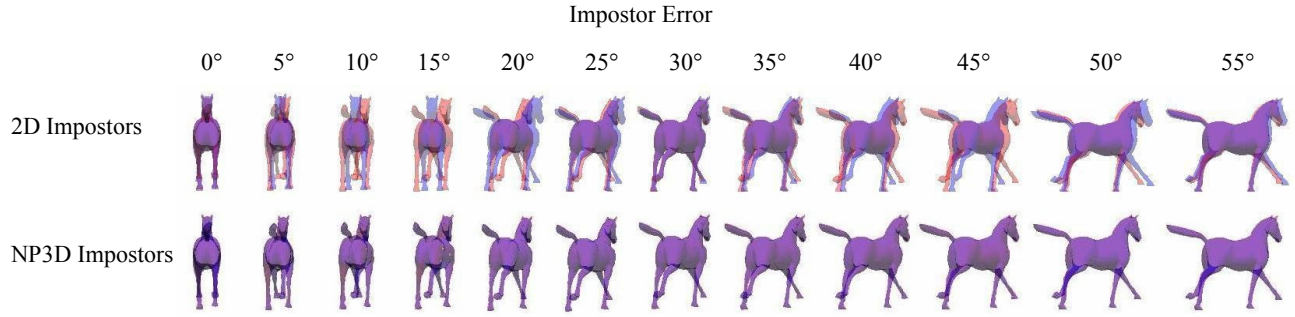

Figure 4. The impostors (shown in blue) are superimposed over the original meshes (shown in red). Screen captures were taken at 5° increments around the horse. The purple portions show where the impostor and original mesh match up. Notice that the 3D impostors are always aligned with the original mesh, whereas the 2D impostors are seldom aligned properly, resulting in animation popping artifacts. All impostors were generated using 30° angular separation and 256x256 pixel texture images.

3. Methods

Our NP3D method is based on existing impostor methods. In the following sections, 2D impostors are reviewed, followed by an explanation of our NP3D impostor method, rendering, and implementation details.

3.1. 2D Impostors

A 2D impostor is created by rendering one quadrilateral projection plane that directly faces the camera (i.e., the plane is perpendicular to the line of sight from the camera to the plane). The appropriate impostor image is retrieved from memory and texture mapped onto the projection plane. See figure 2. During animation, the projection plane rotates and the impostor image changes as needed for the current view.

3.2. NP3D Impostors

NP3D impostors are created in much the same manner as 2D impostors, except that the images are texture mapped onto a simplified version of the original mesh instead of a simple quadrilateral. The simplified meshes contain the entire geometry, including the back-facing polygons, allowing us to reuse the same mesh in all views. Like 2D impostors, only the camera-facing side of the mesh needs to be textured. See figure 3. Also like 2D impostors, the impostor image changes as needed for the current view.

The number of impostor images plays a crucial role in the quality of the animation. For both 2D and NP3D impostors, images are captured at discrete locations around the original mesh. These locations are distanced by a predetermined degree of angular separation. If the angular separation is too small, there will be a lot of images to store and memory will fill up. If the angular separation is too large, popping artifacts will be visible during animation. These artifacts occur when the desired viewing angle differs from the captured viewing angle. In figure 4, a 30° angular separation was used to show the effects of a large angular separation. The 2D impostors do not align with the original meshes, resulting in popping during animation. However the NP3D impostors align perfectly with the original meshes, resulting in zero popping.

3.3. Rendering

We employ LOD techniques to enable efficient rendering of each animation frame. The method utilizes two levels-of-detail to provide realistic, real-time simulations. For high LOD, the full meshes are displayed. For low LOD, the NP3D impostors are created and displayed. To determine the LOD at each time step, we check the pixel-to-texel ratio of each character. If a character's pixel-to-texel ratio is greater than one, we render the character using high LOD. Otherwise the character is rendered using low LOD.

To create the NP3D impostors, the appropriate low-resolution mesh, imposter image, and texture coordinates are retrieved from memory. The low-resolution mesh is chosen such that it corresponds to the character's current animation frame. While the meshes are view-independent, the impostor image and texture coordinates are view-dependent. These are chosen such that they are closest to the desired viewing angle, taking into consideration the horse's rotation and camera viewing angle with respect to the horse. Once we have the appropriate mesh, image, and coordinates, the image is texture mapped onto the mesh using the texture coordinates.

3.4. Implementation

We start with twelve high-resolution 3D horse meshes. The meshes form a gallop animation sequence for a single horse. The meshes have approximately 17,000 polygons each. These meshes were taken from Sumner's horse data set [13]. We use these high-resolution meshes as our high

LOD data set.

We derive our low LOD data set from the high-resolution meshes. The low LOD data set consists of low-resolution meshes, high-detail impostor images, and texture coordinates that map the impostor images onto the low-resolution meshes.

To obtain the low-resolution meshes, we simplify the high-resolution meshes down to 400 polygons each, using Garland's quadrics-based mesh simplification tool [14]. These low-resolution meshes serve as the projection surface for the NP3D impostors.

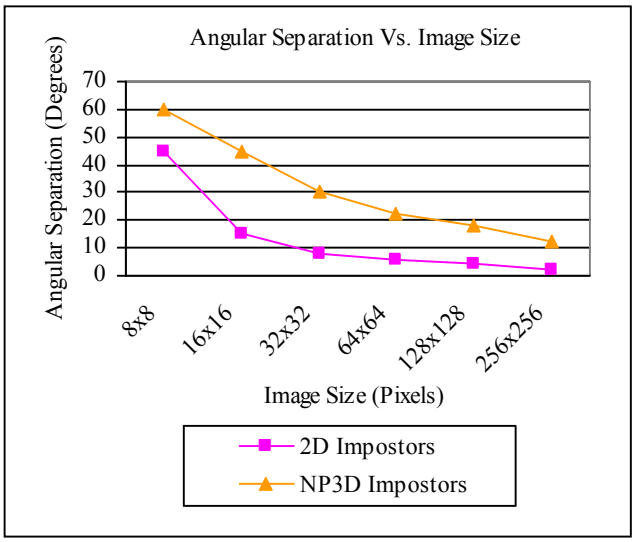

Figure 5. This graph shows the maximum angular separation that is allowable to achieve imperceptible error for different impostor image sizes. At any particular image size, 3D impostors can achieve this goal with greater angular separation and therefore less texture images than 2D impostors.

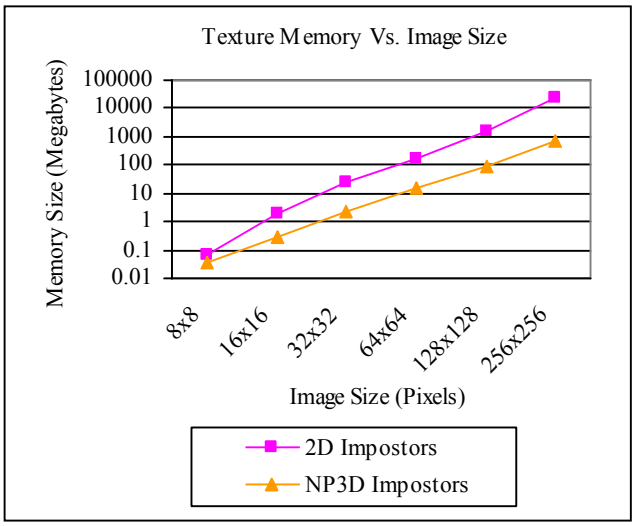

Figure 6. This graph shows the total texture memory size that is necessary to achieve imperceptible error for each of the impostor image sizes. As the image size increases, 3D impostors can achieve this goal with substantially less memory than 2D impostors.

To obtain the high-detail impostor images, we render each high-resolution mesh in a fixed-sized frame buffer. We capture impostor images by longitudinally and latitudinally rotating the camera around the horse at a predetermined degree of angular separation. At each position the frame buffer is read and stored as a texture image. This gives us our NP3D imposter images, which are stored in texture memory. We employ NP3D imposters in order to minimize texture memory consumption, since the imposter must remain in memory at all times.

To obtain the texture coordinates, we render the low-resolution meshes in the same fixed-size frame buffer and with the same camera orientations as in the previous step. But instead of capturing images, we capture the positions of each mesh vertex in the frame buffer. This gives us our texture coordinates, which are used to map the impostor image onto the low-resolution mesh.

Our system randomly seeds the environment with horses at various animation time steps. In each time step, we update each character's current animation frame and randomly move the horse forward, left or right. We employ a simple occupancy grid to prevent collisions between characters.

4. Results

In figure 1, we display three images from our simulation system. The left image shows the crowd simulation without any LOD optimization. The middle image shows the crowd simulation using the traditional 2D impostor method. The right image shows the crowd simulation using our NP3D impostor method. Notice that the NP3D image quality is identical in all cases, but the texture memory utilization and frame rate are substantially better using NP3D impostors. Animated results are provided in the accompanying video.

We quantified the degree to which NP3D impostors allow for the reduction of angular separation of texture images. We initially tried to use an objective measure such as pixel intensity error, but found that it correlated poorly with human perceptual error. In order to insure that our results are perceptually meaningful, a human user identified the greatest angular separations at which rendering artifacts are not visible. This was done by gradually increasing the angular separation, rotating the character, and visually comparing the impostor to the original mesh. This experiment was repeated for different image sizes, ranging from 256x256 down to 8x8 pixels. See figure 5. Note that NP3D impostors allow for significantly greater degrees of angular separation at every size.

Based upon these required angular separations, we calculated and compared the required amount of texture

memory for each method. See figure 6. NP3D impostors consume substantially less memory than 2D impostors while maintaining the same visual quality. This is possible because NP3D impostors allow for a larger angular separation between impostor images and thus require fewer images. The NP3D method is able to fit all images into the memory available on current computer systems. However for 128x128 and 256x256 sized images, the 2D requirement far exceeds the available memory.

We performed another experiment to see how our NP3D impostor method compares to simple mesh simplification LOD techniques without texture mapping. In general mesh simplification produces noticeably bad renderings when meshes are over simplified. See figure 7. Our human user

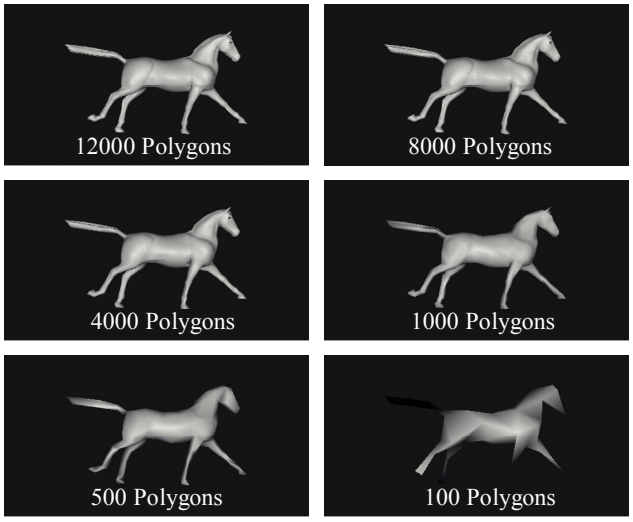

Figure 7. Simplified meshes without texture mapping contain noticeable artifacts.

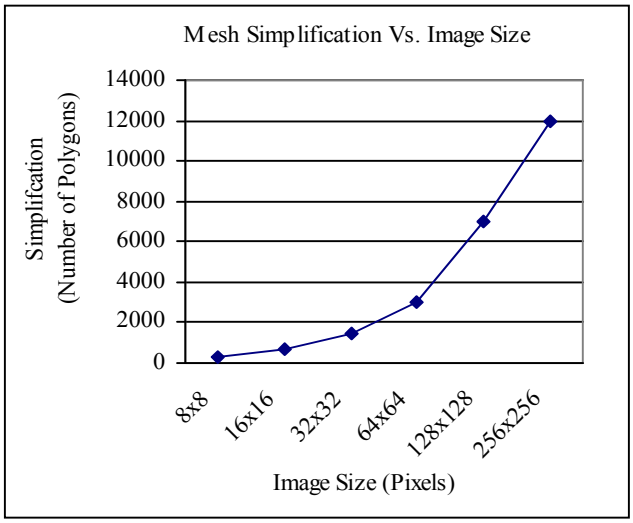

Figure 8. This graph shows the minimum number of polygons that are necessary in order to achieve imperceptible errors for various image sizes. Larger images show more detail and require more polygons to retain muscle tone and facial details of the original meshes.

determined the smallest number of polygons at which rendering artifacts are not visible for each image size. See figure 8. As the image size increases, more polygons are necessary to achieve zero rendering artifacts.

Using these determined polygon simplifications, we created a simulation using mesh simplification. We then compared the crowd size capabilities of the simulation using original full meshes only, mesh simplification, 2D impostors, and NP3D impostors. See figure 9. Overall the NP3D impostor method surpasses the capability of the other methods. It is able to animate over 900 characters with 256x256 sized images. The 2D and NP3D impostor methods followed each other closely through the 64x64 sized images, but the 2D impostor method could not render any horses when the image size reaches or exceeds 128x128, because the required memory exceeds the system's resources.

To better understand the texture memory constraint, we reran the previous comparison using only a side-view of the simulation. This allowed us to strip the system of all images, except those taken at 0° latitude. This reduced the texture consumption, so that the 2D impostor simulation could run. See figure 10. Again the NP3D impostor method exceeds the capability of the 2D method. The 2D method drops off starting with the 128x128 image size, due to texture swapping between the graphics card memory and main memory.

We tested mesh simplification in two ways. First we used two levels-of-detail with transition at the same depth as the imposter methods. With mesh simplification, there is a tradeoff between using fewer polygons with a distant transition depth and having a closer transition depth with more polygons in the simplified mesh. Note that the optimal tradeoff occurs with the depth plane equivalent to a 64x64 image size. It is possible to achieve better performance using multiple levels-of-detail. We created 7 depth zones and used the optimal number of polygons in each zone. In the figures 9 and 10, the single lone dot represents the largest crowd size if multiple levels-of-detail are used with mesh simplification. While this exceeds the performance of mesh simplification with a single transition plane, it does not perform as well as NP3D impostors.

We verified that all simulations were limited either by polygon count or texture memory. In the case of our test machine, the limits are 30 million polygons per second and 1 GB of texture memory.

5. Conclusions

We demonstrated how to animate and render hundreds of characters in real-time. We developed an algorithm that combines high-resolution 3D scanned meshes and synthesized NP3D impostors into a new rendering optimization strategy for crowd simulation. This system

allows the characters to rotate with respect to the camera position without animation artifacts. It also reduces the necessary memory requirements. Our method is ideal for video game and interactive rendering, because it allows for real-time rendering.

References

[1] Hoppe, H., Progressive meshes. In Proceedings of the 23rd Annual Conference on Computer Graphics and interactive Techniques SIGGRAPH '96. ACM Press, New York, NY, pp. 99-108, 1996.

[2] Aubel, A., Boulic, R., Thalmann, D., Real-time display of virtual humans: levels of details and impostors. IEEE Transactions on Circuits and Systems for Video Technology, Vol. 10, No. 2, 207–217, March 2000.

[3] Tecchia, F., Loscos, D., Chrysanthou, Y., Image-Based Crowd Rendering, IEEE Computer Graphics and Applications, pp. 36-43, 2002.

[4] O'Sullivan, C., Cassell, J., Vilhjalmsson, H., Dingliana, J., Dobbyn, S., McNamee, B., Peters, C., Giang, T., Levels of Detail for Crowds and Groups. Computer Graphics Forum, Vol. 21, No. 4, 733–741, 2002.

[5] Ahn, J., Kwangyun, K., Motion Level-of-Detail: A Simplification Method on Crowd Scene. Proceedings Computer Animation and Social Agents, 2004.

[6] Ahn, J., Oh, S., and Wohn, K. 2006. Optimized motion simplification for crowd animation: Research Articles. Computer Animation and Virtual Worlds, Vol. 17, No. 3-4, 155-165, July 2006.

[7] Dobbyn, S., Hamill, J., O'Conor, K., O'Sullivan, C., Geopostors: a real-time geometry / impostor crowd rendering system. Symposium on interactive 3D Graphics and Games, I3D '05. ACM Press, New York, NY, 95-102, 2005.

[8] Wand, M., Straber, W., Multi-resolution rendering of complex animated scenes, Eurographics Association, Vol. 21, No. 3, September 2002.

[9] Debevec, P. E., Taylor, C. J., Malik, J., Modeling and rendering architecture from photographs: a hybrid geometry- and image-based approach. Computer Graphics and Interactive Techniques SIGGRAPH '96. ACM Press, New York, NY, 11-20, 1996.

[10] Sillion, F., Drettakis, G., Bodelet, B., Efficient Impostor Manipulation for Real-Time Visualization of Urban Scenery, Computer Graphics Forum, Vol. 16, No, 3, C207–C218, 1997.

[11] Decoret, X., Sillion, F., Schaufler, G., Dorsey, J., Multi-layered impostors for accelerated rendering, Computer Graphics Forum, Vol. 18, No, 3, 61–73, 1999.

[12] Schaufler, G., Dynamically generated impostors. In GI Workshop on Modeling, Virtual Worlds, (Nov. 1995), Fellner D. W., (Ed.), pp. 129–135.

[13] Sumner, R. W. and Popović, J. 2004. Deformation transfer for triangle meshes. ACM Trans. Graph. 23, 3 (Aug. 2004), 399-405.

[14] Garland, M. and Heckbert, P. S. 1997. Surface simplification using quadric error metrics. Computer Graphics and Interactive Techniques. ACM Press/Addison-Wesley Publishing Co., New York, NY, 209-216.

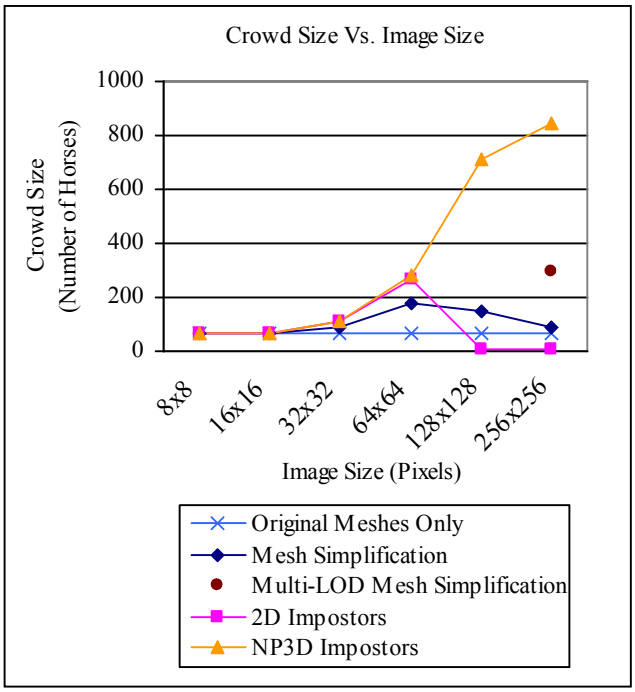

Figure 9. This graph shows the maximum number of horses that each method can generated while maintaining an animation rate of at least 30 frames per second. Of all methods, our 3D impostor method can generate the most number of horses in real-time.

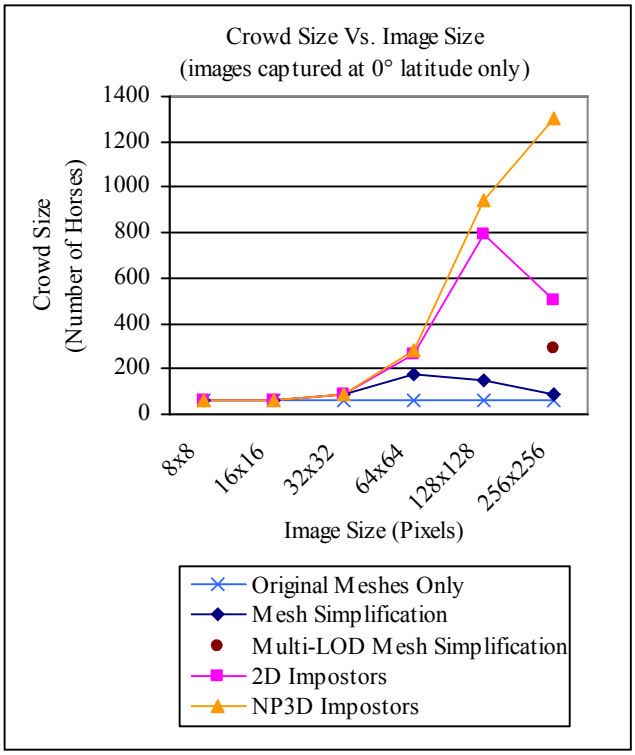

Figure 10. This graph shows the same measurements at the previous one, except it shows what happens if images are captured only around 0° latitude. The 2D impostor method drops off starting all 128x128 due to texture memory swapping.