SHADOW MAPPING

Karthik Nallapeta Subramanya
USC ID: 9234-3900-71
Email: knsubram@usc.edu

Izaaz Yunus Hasi Ebrahim
USC ID: 4680-7077-72
Email: yunushas@usc.edu

Sourav Dey
USC ID: 2232-2171-84
Email: souravde@usc.edu

Mehul Mittal
USC ID: 6167-6540-13
Email: mehulmit@usc.edu

ABSTRACT
Shadow Mapping is particularly amenable to hardware implementation because it makes use of existing hardware functionality – texturing and depth buffering. The extra burden it places on hardware is the need to perform a high-precision scalar comparison for each pixel fetched from shadow map texture.

Keywords
Shadows, Z-Buffering, Square Directional Lights, Light Buffer.

1. INTRODUCTION
Shadow mapping is a process by which shadows are added to 3D computer graphics. A shadow makes 3D computer graphics look better. A scene without a shadow looks unreal. They add a sense of depth to the scene, but shadows come at a cost, computational cost. The trouble is that shadows require a visibility test for each light source. This paper will focus on a simple shadow mapping technique used by us on the graphics library which we developed during the semester.

The display of shadows may make the shape and relative position of objects in computer generated scenes more comprehensible. Shadows emphasize and may serve to clarify the three dimensional nature of the forms displayed. The shadows cast by a point source of light onto a flat surface represent, like a perspective transformation, a projection of the scene onto a plane. This simplified situation offers a convenient way of understanding the information that shadows convey. A scene rendered with shadows contains two views in one image.

Shadow mapping is an image-based shadowing technique developed by Lance Williams [1] in 1978. It is particularly amenable to hardware implementation because it makes use of existing hardware functionality – texturing and depth buffering. The only extra burden it places on hardware is the need to perform a high-precision scalar comparison for each texel fetched from the shadow map texture. Shadow maps are also attractive to application programmers because they are very easy to use, and unlike stenciled shadow volumes, they require no additional geometry processing. [2]

2. DESCRIPTION
Shadow mapping checks if a point on a surface is visible from the light. If a point is visible from the light then it's obviously not in shadow, otherwise it is. The basic shadow mapping algorithm can be described as short as this:

1. Render the scene from the lights view and store the depths as shadow map
2. Render the scene from the camera and compare the depths, if the current fragments depth is greater than the shadow depth then the fragment is in shadow

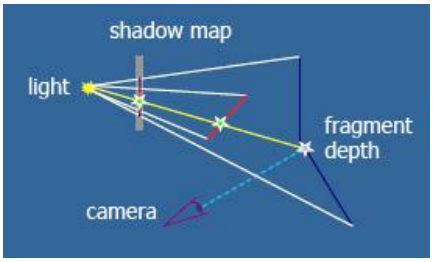

Figure 1. Concept of shadow mapping

The two big problem areas with shadow mapping:

- Hard to select an appropriate bias (epsilon)
- Hard to get rid of artifacts at shadow edges

The clever insight of shadow mapping is that the depth buffer generated by rendering the scene from the light's point of view is a pre-computed light visibility test over the light's view volume. The light's depth buffer (a.k.a. the shadow map) partitions the view volume of the light into two regions: the shadowed region and the unshadowed region. The visibility test is of the form where p is a point in the light's image space.[2]

$$p_z \leq shadow\_map(p_x, p_y)$$

2.1 GRAPHICS LIBRARY USED
There were a lot of options to the graphics library that we could use and implement shadow mapping. We thought of starting with OpenGL libraries and integrate the FLTK toolkit along with it to compute shadow buffers using the concept of ray tracing.

The other option we had was to use the graphics rendering library which we built during the period of 2 months in our 3D computer graphics class. We chose to work with this as we had more control over the structures and we could learn how the lighting equation is computed to its finest details.

We started without texturing and anti-aliasing. Then the project was extended to include anti-aliasing to give smother shadow.

Also texturing was added, so that shadows are visible even if texturing is applied on the objects.

2.2 LIGHT STRUCTURE

The light source has been modified. The new light sources have few extra parameters.

- Position
- LookAt
- WorldUp
- LightZBuffer
- FOV
- CastShadows

CastShadows is a boolean parameter which is used to decide if the particular light will cast shadow or not. This parameter is used to have a control over the light source. It lets you decide if the light will cast shadows or not, hence limiting the computational cost.

LightZBuffer is a floating point array which is used to store the shadow Z values computed from the lights point of view.

Position specifies the position where the light is placed. LookAt specifies the direction of the light. WorldUp is used to compute the lighting equation while computing LightZBuffer for each light.

The light source does not implement intensity fall-off over distance. Hence the intensity of the light source on every point on the surface that it illuminates is constant.

3. ARCHITECTURAL CHANGES

Each object is converted from the model space to the screen space and goes through the transition shown in the figure 1.

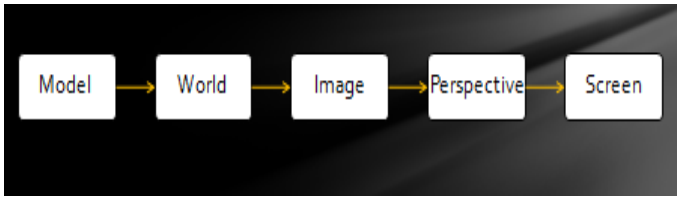

Figure 2. Original space

Each rendered triangle of an object goes through the following conversion as shown in the figure 1. Initially the object is present in the model space which is converted into world space. These world space coordinates are converted into the camera space also called as the image space. These image space coordinates represent the scene from the camera's point of view and decides the orientation of the scene. These are then converted into a perspective space coordinate which bounds the X and Y values in the range of 0 to 1 and Z values are bounded by Z_{MAX}/d, where d is calculated using the FOV. Finally, these triangles are converted into screen space, thus having the resolution of the screen. The triangles are then interpolated and rendered for each pixel that it covers.

We have added a few more spaces to the original existing model space. These spaces are called Light space and Light Buffer space. The light space is calculated for each light. The object after being converted into a world space is then converted into a light space. Light space is used to represent the scene from the light point of view. Then these light space coordinates are converted into the Light Buffer space which be said as the equivalent of the screen space in the original model. The coordinates at Light Buffer space is used to calculate the LightZBuffer that is the Z values as seen from the lights point of view. This new model space is represented in figure 2.

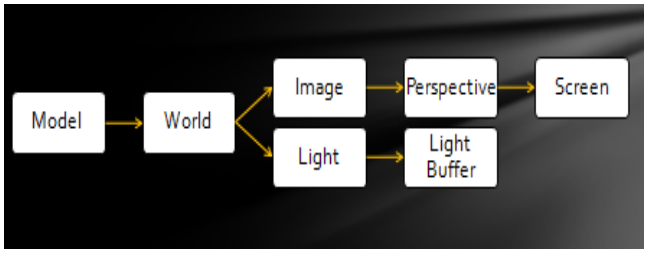

Figure 3. Modified space

Finally the model space which we have used for the shadow mapping technique is shown in figure 3. In this space we get back to the world space from the screen space. This will be useful for computing the lighting calculation which will be explained in the later phase of the paper.

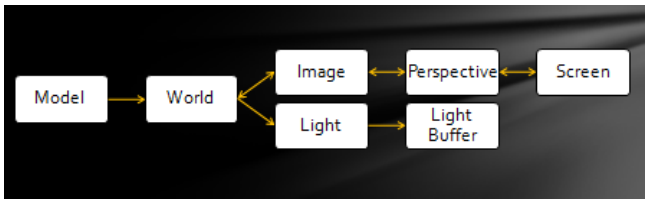

Figure 4. Final spaces

4. ALGORITHM

1. Populate LightZBuffer for each light who's CastShadows is set to true before rendering.
2. While rendering (For each pixel), compare LightZBuffer with the original Z value.
3. If the values differ by a particular threshold, then compute shadow calculation
4. Else continue

Old Code

```
while(!EOF)
{
   putTriangle()
}
```

New Code

```
while(!EOF)
{
   populateShadowBuf()
}

while(!EOF)
{
   putTriangle()
}
```

Figure 4. Changes in the renderer.

Each light, directional light is manipulated by treating it like a camera. Each light in the scene has a z-buffer which is a two dimensional array. The functionality of this z-buffer is the same as

the camera z-buffer. The LightZBuffer is a two dimensional array. The integer x and y are the respective indices of the array. Each (x, y) space in the light buffer refers to the z value visible at that (x, y) point. Before rendering, the light buffers are first populated using the same triangles. The steps to do the same are similar to the rendering process. They are depicted below described below.

Light Sequence:
Model space → World Space → Light Space → Light Buffer Space

1. Use Shadow Buffers in Shading Calculations

Our method has been implemented for Phong Shading. Phong shading computes the shading effect on every pixel of a triangle from each of the lights in the scene. Our method deviates from convention here.
Before computing and adding the effect of a light on a pixel, we test if the pixel is visible from every light source.

This is done as follows for each light source:

Rendering Screen Coordinate → Rendering Perspective Coordinate → Rendering Camera Coordinate → World Coordinate → Light Space Coordinate → Light Buffer Coordinate

The LB(x, y, z) coordinate that is obtained from the conversion to light buffer coordinate need not be integer values. The light buffer is stored as (x, y) indices. Hence the light buffer coordinates (x, y) are mapped to the nearest integer (x, y) values. These integer values are used to find the z coordinate existing in the light buffer (existingZ).

If LB.z is within a threshold of existingZ, then the point is affected by the light source and hence will be used in the shading equation of the pixel in the rendering process.

5. EFFICIENCY

The time taken for completely rendering a scene is directly proportional to the no of light sources casting shadows. Each light sources takes requires a LigthZBuffer which is computed before the scene is actually rendered. Each LightZBuffer takes processing time which is equal to the rendering time. Therefore more the no of lights casting shadow more is the computational time. Also each light has additional space requirements as a LightZBuffer is added for each light.
Time – Since the computations in Step 1 are similar to the rendering process computation, we can say that the time taken for computation of the shadow is the number of lights multiplied by the time taken for running each the rendering process once.
Space – The resolution of a shadow buffer determines the quality of the shadow. The quality of the shadow is better for higher resolutions of the shadow buffer.

6. RESULTS

The results of the experiment can be seen in the images shown. There are 3 light sources in the scene. One of them is casting a shadow. The other two light sources are located infinitely far and illuminating the scene in a direction. We have shown the shodow of the teapot from 6 different light parameters. The different light paramers are chosen to test the depth of the shadow. Also there is a shadow of the lid on top of the teapot itself.

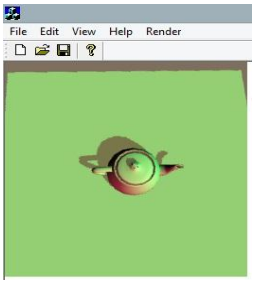

Figure 5

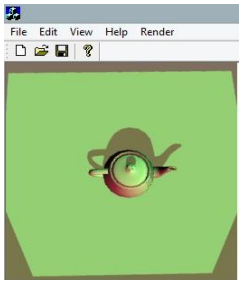

Figure 6

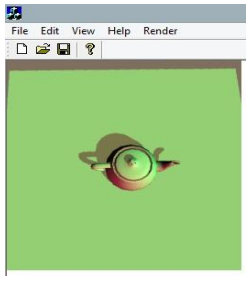

Figure 7

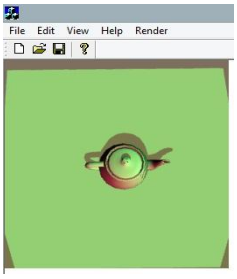

Figure 8

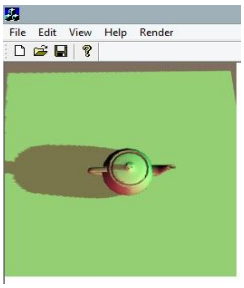

Figure 9

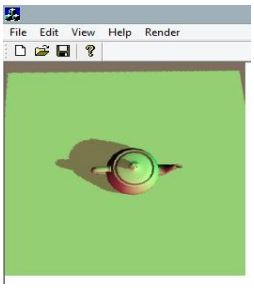

Figure 10

7. ACKNOWLEDGMENTS

The authors would like to thank Prof. Neumann, Ulrich for granting us the project, Shadow Mapping.

8. REFERENCES

[1] Lance Williams 1976. Casting Curved Shadows on Curved Surfaces. http://artis.imag.fr/Members/Cyril.Soler/DEA/Ombres/Papers/William.Sig78.pdf

[2] Cass Everitt, Ashu Rege, Cem Cebenoyan. Hardware Shadow Mapping. http://www.cs.berkeley.edu/~ravir/6160/papers/shadow_mapping.pdf

[3] Randima Fernando. Percentage-Closer Soft Shadows. http://developer.download.nvidia.com/shaderlibrary/docs/shadow_PCSS.pdf

[4] Cascaded Shadow Maps 2008, NVidia 2008

[5] http://en.wikipedia.org/wiki/Shadow_mapping