Computer Graphics Continued

Definitions

- **Stencil buffer** - another memory buffer similar to the frame buffer and Z-buffer. It is used to identify areas of the frame buffer as drawable or not drawable. This allows parts of the framebuffer to be updated and other parts to be masked.
- **Global illumination** - models which consider light reflected from objects in addition to light sources. They tend to look more realistic than images rendered using only **direct illumination** (light from light sources).

Shadows

- The shading models (ie. Gouraud, Phong, Blinn-Phong) can only brighten an object. They add to the intensity.
- To create shadows it is necessary to add a mechanism which determines when light is blocked.
- Two popular method to generate realistic shadows are shadow volumes and shadow maps. Both have been used in computer games.

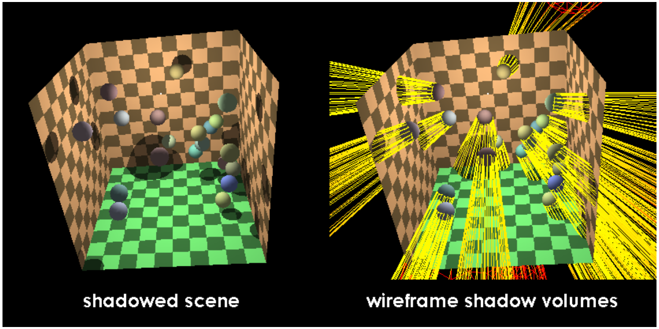
Shadow Volumes

- This method works by creating three dimensional volumes in the scene which represent areas which are in shadow.
- Polygons which fall into these volumes are rendered as though they were in the shadow of an object. Objects not in these volumes are rendered normally.

Shadow Volumes Algorithm

- Rays are cast from the light source through each vertex in the objects which will create a shadow.
- For polygonal models, the silhouette edges are used to create the volume. Silhouette edges are determined by finding polygons which share an edge and where one polygon faces the light and the other faces away from the light.
- The silhouette edges are extended away from the light source along the rays emanating from it.
- The front and back of the volume may be capped depending on the technique being used.

Shadow Volumes Algorithm



Shadow volumes can be seen extending away from the light source in the right image.

Rendering with Shadow Volumes

- The rendering algorithm involves three steps:
  1. First render the scene as if it were completely in shadow.
  2. Using the depth information for the scene, from the Z-buffer, determine which areas of the scene are in shadow. These areas should be masked using a stencil buffer so they cannot be modified.
  3. Render the scene a second time with normal lighting but use the stencil buffer to mask the shadowed areas.
- For multiple light sources, the second and third steps are repeated. This requires the lit areas to be blended with the existing image in step three instead of completely replacing the previously rendered image.
- There are several variations of step 2 which trade accuracy for performance.

Shadow Volumes

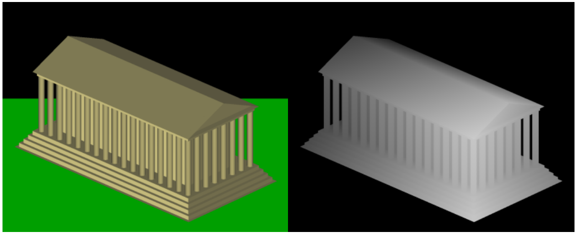
- Generally create highly accurate shadows.
- Shadows along the silhouette edge which an object casts upon itself (self shadowing) may be too sharp which will reveal the polygonal structure of the model.
- They require a considerable amount of rendering time.
- Objects with many edges can create complex volumes which require a lot of processing time to calculate.

Shadow Mapping

- Shadow mapping works by rendering an image from the light's viewpoint (instead of the viewer) and assuming that anything which isn't visible to the light will be in shadow.
- The algorithm involves two steps:
  1. creating the shadow map
  2. shading the scene
- If more than one light is used then multiple shading steps will be used.

Creating the Shadow Map

- This involves calculating the depth from the light to objects in the scene and storing the values in a depth buffer. The depth buffer is the shadow mask.
- The depth buffer can be reused until the light or objects in the scene are moved. The viewpoint can be moved without updating the depth buffer.
- Depth buffers must be calculated for each light source.

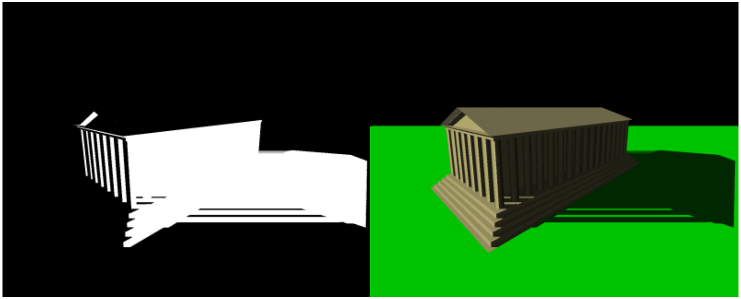


The left image is the view from the light source. The right image is the depth buffer using the light source as the viewpoint. Anything behind the object will be in shadow.

Shading the Scene Using a Shadow Map

- There are three steps to rendering the scene:
  1. The position of objects in the scene is transformed as though it were being viewed from the light source (transformed into light space coordinates). This means that the Z value of the transformed object will indicate the distance from the light source.
  2. The transformed object's depth coordinates are compared to the depth value in the depth buffer. If the object's Z value is less than the value in the buffer then it is in the light, if it is greater than the value in the depth buffer then it is in shadow.
  3. The scene is rendered from the viewpoint with areas in shadow receiving only ambient light and other areas receiving all light. Multiple rendering passes may be used to render the entire scene first in shadow and then add the lighted areas.

Shading the Scene Using a Shadow Map



The left image shows the areas which will be drawn as shadowed.
The right image is the final image with shadows.

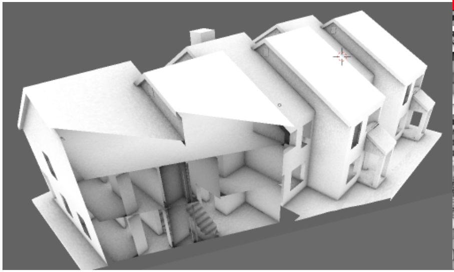
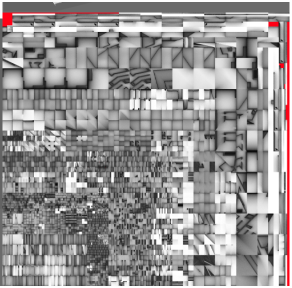
Shadow Mapping

- Shadow mapping is often faster than shadow volumes but it tends to be less accurate.
- The edges of shadows can appear rough (stepped) if the shadow map doesn't have enough resolution.
- They require a depth buffer to store the shadow map.
- They can be used to create soft edged shadows.

Light Maps

- Light maps are used to precalculate lighting and store it in a texture map. This allows for complex lighting to be calculated off-line.
- Textures normally store the colour of the object but there is no reason they cannot store lighting information as well.
- The object is rendered using any illumination model. This can include complex and computationally expensive techniques such as ray-tracing or radiosity. The appearance of the rendered object is stored in a texture which is used to light the model in real time.
- The lighting texture can include all colour information for the object or it can store only the lighting information which is used to modulate the illumination.
- The main drawback to this method is that the lighting is static.

Lightmap Example

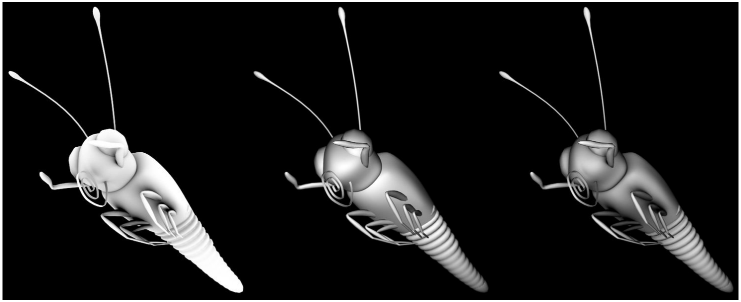



Left is the object with precalculated lighting. Right is the lightmap stored as a texture.

Ambient Occlusion

- Ambient occlusion is a method which approximates global illumination.
- It operates by casting rays from points on the object to be illuminated. Rays can either intersect with the object itself or reach the “sky.”
- The brightness of the points on the object depends on the number of rays which reach the sky. If many rays do not intersect with the object then that point will be lighter than if many rays do intersect with it.
- Points surrounded by a lot of geometry tend to be darker than points in open spaces.
- It allows the viewer to more easily perceive the shape of the object.
- The results of the ambient occlusion calculations can be stored as a texture map or can be used to adjust the brightness of the object’s vertices.

Ambient Occlusion Example



Left: object rendered with ambient occlusion. Middle: diffuse lighting. Right: Ambient occlusion and diffuse lighting combined.

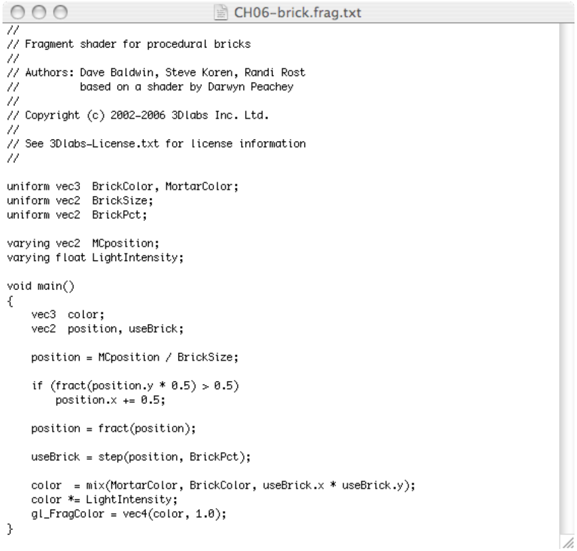
Programmable Shading Languages

- Programming languages used to control parts of a graphics system.
- Originally used for rendering high quality images but not in real time. Pixar uses the Renderman language for cinema quality animation.
- Prior to their being adapted for real time, consumer applications programming using graphics libraries primarily involved setting parameters and letting the library perform built in static routines. Function calls were used to set the parameters, such as: glShadeModel(GL_SMOOTH), glPolygonMode(GL_FRONT_AND_BACK, GL_FILL) glRotatef (270.0, 1.0, 0.0, 0.0) glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, white).

Programmable Shading Languages

- Shading languages found in modern game systems allow the graphics programmer to write the routines which control the graphics hardware. This allows more control over the operations performed by the graphics accelerator.
- Video cards (graphics accelerator cards) have reached the point where they are high performance, specialized operation processors which often have more processing power than the CPU. These are called GPUs (Graphics Processing Units).
- GPUs are programmable pipelined processors which transform the three dimensional scene information into a rendered image in the framebuffer.
- Early shading languages were similar to assembly language. Modern languages now look more like C.
- The three commonly used shading languages are HLSL used in Direct3D, GLSL used in OpenGL, and Cg which was developed by Nvidia and works in both Direct3D and OpenGL.

Shader Program Example



```
//
// Fragment shader for procedural bricks
//
// Authors: Dave Baldwin, Steve Koren, Randi Rost
//          based on a shader by Darwyn Peachey
//
// Copyright (c) 2002-2006 3Dlabs Inc. Ltd.
//
// See 3Dlabs-License.txt for license information
//

uniform vec3  BrickColor, MortarColor;
uniform vec2  BrickSize;
uniform vec2  BrickPct;

varying vec2  MCposition;
varying float LightIntensity;

void main()
{
    vec3  color;
    vec2  position, useBrick;

    position = MCposition / BrickSize;

    if (fract(position.y * 0.5) > 0.5)
        position.x += 0.5;

    position = fract(position);

    useBrick = step(position, BrickPct);

    color  = mix(MortarColor, BrickColor, useBrick.x * useBrick.y);
    color *= LightIntensity;
    gl_FragColor = vec4(color, 1.0);
}
```

Programmable Shading Languages

- Shader programs can be either vertex or fragment (pixel) shaders.
- A fragment is a pixel with its associated data such as window coordinates, depth, colour, texture coordinates.
- Vertex shaders can affect the following elements:
  - position of the object and perspective transformations
  - normal transformation and renormalization
  - texture coordinates
  - per vertex lighting
  - colour of materials
- Fragment shaders can affect the following elements:
  - texture operations
  - colour summing
  - fog

Programmable Shading Languages

- There are normally multiple pipelined texture and fragment processors which run in parallel.
- Inputs and outputs for shaders are well defined. These can be standard graphical elements such as vertex positions or normal vectors or they can also be user defined values.
- Vertices are not created by the shaders. They can be modified or dropped but new ones cannot be manufactured.

Other Uses for Programmable Shading Hardware

- The shading hardware is a special purpose processor used in parallel to the regular CPU. It's architecture which makes it useful for high performance calculations also makes it a good choice for any other repetitive calculations.
- GPUs have been used in physics engines and for interpolation in keyframe animation.
- Work is being done in using the GPU as a general purpose processor, called GPGPU.

References

- Blender Artists Forum, lightmap sample, http://blenderartists.org/forum/showthread.php?p=791818.
- Rost, Randi J., OpenGL Shading Language, 2nd Edition, Addison Wesley, 2006.
- Torque Game Engine 1.5 Demo, www.garagegames.com.
- Wikipedia web site, www.wikipedia.org.