

Homework 08

Rendering 3D Geometry

David C. Banks

Electrical Engineering and Computer Science
University of Tennessee

2009

Textbook

Chapter 8 The Graphics Pipeline Chapter 21 Color

Overview

Write a program **sphereTriangulate**

Store sphere in a file

Write a program **sceneView3D**

Render spheres scenes into test images

Refinement 0, 1, 2, 3, 4

Different radii

Different translations

Write a program **shootem3D**

Store 3D scenes in files

Render 3D scenes into movie

Sphere

Write a program **sphereTriangulate** that generates a sphere

Unit sphere centered at origin

Sphere is composed of triangles

Recursive subdivision of octagon

Top at $(0,0,1)$, bottom at $(0,0,-1)$

4 vertices on x, y axes: $(1,0,0)$ $(0,1,0)$ $(-1,0,0)$ $(0,-1,0)$

sphereTriangulate level 2 > sphere.geom

triangle3D

Create new primitives for 3D geometry

```
triangle3D
```

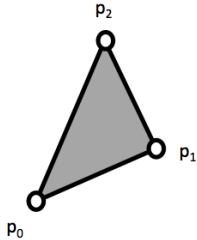
```
{  
vertex3D { x 10.0 y 20.0 z 15.0 }  
normal3D { nx 1.0 ny 0.0 nz 0.0 }  
reflectRGB { r 1.0 g 0.0 b 0.0 }  
...  
}
```

Note: the normal might not be unit length

For a sphere centered at origin, normal = vertex

Recursive subdivision

Subdivide a triangle with vertices p_0 , p_1 , p_2



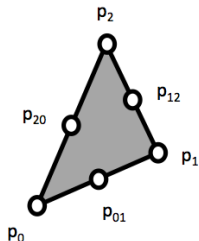
Recursive subdivision

Midpoint of edge is average of endpoints

$$p_{01} = (p_0 + p_1) / 2$$

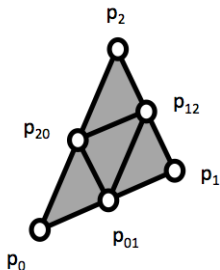
$$p_{12} = (p_1 + p_2) / 2$$

$$p_{20} = (p_2 + p_0) / 2$$



Recursive subdivision

Replace triangle with 4 triangles



Recursive subdivision

Push vertex out onto unit sphere

$p_{\text{OnUnitSphere}} = p / \text{length}(p)$

Keep triangles in a list

Create new list when subdividing triangles

New list has 4 times as many triangles

Particles as dots

Split triangle on unit sphere into 4 subtriangles

```
subdivide ( triangleList L, triangle t)
{
  point3D p01 = (t.p0+t.p1)/2;
  point3D p12 = (t.p1+t.p2)/2;
  point3D p20 = (t.p2+t.p0)/2;
  p01 /= length(p01);
  p12 /= length(p12);
  p20 /= length(p20);
  addToList (L, makeTriangle (p0,   p01, p20));
  addToList (L, makeTriangle (p1,   p12, p01));
  addToList (L, makeTriangle (p2,   p20, p12));
  addToList (L, makeTriangle (p01,  p12, p20));
}
```

SceneView3D

Write a program **sceneView3D** to read scene, display it

Coordinates: x right; y up; z toward viewer

Scene contents have negative z values

Cull any triangle “behind” the $z=0$ plane

Matrix transform

Multiply matrix stack to yield active matrix M

Append homogeneous coordinate to vertices

That is, (x,y,z) becomes $(x,y,z,1)$

Apply M to homogeneous vertices in geometry

Divide by homogeneous coordinate and project

That is, (x,y,z,h) becomes $(x/h,y/h)$

Fill triangle

Depth buffer

Image buffer includes red, green, blue

Determine color at each vertex

Modify triangle renderer to interpolate r,g,b

Output using ppm format

P3

3 2

255

100 200 210 100 200 210 100 200 210

100 200 210 100 200 210 100 200 210

This is a milestone: filled triangle from 3D file format

Depth buffer

Image buffer includes depth z

Modify triangle renderer to interpolate z

Initialize buffer to “-infinity”

If new depth z is closer than old, replace pixel

This is a milestone: resolve interpenetrating triangles

Diffuse reflection

Diffuse reflection at a vertex x of triangle

Reflection from a single light in unit direction L

Light has intensity (light.red, light.green, light.blue)

Point x has diffuse reflectance (x.red, x.green, x.blue)

Point x has unit normal N

Diffuse reflection is light * reflectance * $N \cdot L$ (if $L \cdot N \geq 0$)

Otherwise, clamp reflected light to 0 (for opaque surface)

Or, negate to make value positive

This is a milestone: local illumination

Reflection

General description for 1 light: $\mathcal{L}_{out} = \mathcal{L}_{in} f(v_{in}, v_{out}) N \cdot L$
“Scattering kernel” $\mathcal{L}_{in} f(v_{in}, v_{out}) N \cdot L$

Diffuse reflection

For multiple lights, add the scattering kernel multiple times

$\mathcal{L}_{in}[i]$ is i^{th} light source

Compute sum $\sum_{i=1}^{\text{numLights}} \mathcal{L}_{in}[i] f(v_{in}, v_{out}) N \cdot L$

This is a milestone: multiple lights

Visibility

What if light is hidden by intervening geometry?

Use geometry term $g[i] = 1$ if light is visible; otherwise, 0

Compute sum $\sum_{i=1}^{\text{numLights}} \mathcal{L}_{in}[i] f(v_{in}, v_{out}) N \cdot L g[i]$

Ray-triangle intersection

How to determining geometry (visibility) term $g[i]$?

Intersect ray from x to light with the scene

Could be a lot of ray-triangle intersection tests

Most of the tests will return “false”

How to locate ray-triangle intersection?

Search web for ray triangle intersection

This is a milestone: shadows

Reflectance

Simple diffuse reflectance model

Initialize vertex $\text{refl.red} = \text{refl.blue} = \text{refl.green} = 0.0$

At vertex, loop over all point lights in scene

Trace ray to each point light

If another triangle lies between, continue; otherwise

Compute unit length light vector L

If $L \cdot N < 0$, continue; otherwise

$\text{refl.red} += L \cdot N * \text{light.red} * x.\text{red}$

$\text{refl.green} += L \cdot N * \text{light.green} * x.\text{green}$

$\text{refl.blue} += L \cdot N * \text{light.blue} * x.\text{blue}$

Transforming the normal

Matrix M is product of matrix stack

Transform each vertex x to Mx

M is a homogeneous matrix

Let \hat{M} be the 3x3 non-homogeneous submatrix

Transform each normal n to $(\hat{M}^{-1})^T n$

Transpose of inverse of \hat{M}

Note: if M is a translation, $\hat{M} = I$, $(\hat{M}^{-1})^T = \hat{M}$

Note: if M is a rotation, $(\hat{M}^{-1})^T = \hat{M}$

Point light

Create new primitives for 3D point light

```
pointLight3D  
{  
  position3D { x 50.0 y 30.0 z 10.0 }  
  emittanceRGB { r 1.0 g 1.0 b 1.0 }  
}
```

Scene description

SCENE version 4.0

dimension 3

scene

{

transform

{

matrix3DH

{

m00 1.0 m01 0.0 m02 0.0 m03 9.0

m10 0.0 m11 1.0 m12 0.0 m13 5.0

m20 0.0 m21 0.0 m22 1.0 m23 1.0

m30 0.0 m31 0.0 m32 0.0 m33 1.0

}

Scene description

(continued)

matrix3DH

```
{  
  m00 5.0  m01 0.0  m02 0.0  m03 0.0  
  m10 0.0  m11 5.0  m12 0.0  m13 0.0  
  m20 0.0  m21 0.0  m22 5.0  m23 0.0  
  m30 0.0  m31 0.0  m32 0.0  m33 1.0  
}
```

Scene description

```
(continued)  
geometry ...  
}
```

Shootem3D

Write a program **shootem3D** to shoot spheres from a sphere
Random direction, velocity, radius
Re-use sphere geometry from a file
Use homogeneous matrix to translate
Use homogeneous matrix to scale radius
Write scene to output file