

How to Freeze and Unfreeze an Animation and Maintain Time Continuity

1



Oregon State

University

Mike Bailey

mjb@cs.oregonstate.edu



This work is licensed under a [Creative Commons](#)
[Attribution-NonCommercial-NoDerivatives 4.0](#)
[International License](#)



Oregon State
University
Computer Graphics

FreezeAnimation.pptx

mjb – November 5, 2023

1

2

At the Top of the Program:

```
const int MS_PER_CYCLE = 10000;           // 10000 milliseconds = 10 seconds

float    TimeFrozen;                    // when animation was frozen
float    TimeUnfrozen;                  // when animation was unfrozen
float    TimeElapsed;                  // how much time elapsed between freezing and unfreezing
```

In Reset():

```
TimeElapsed = 0.f;
```



Oregon State
University
Computer Graphics

mjb – November 5, 2023

2

1

In Keyboard()

3

```
case 'F':  
case 'f':  
    Freeze = ! Freeze;  
    if( Freeze )  
    {  
        glutIdleFunc(NULL);  
        TimeFrozen = Time - TimeElapsed;  
        if( TimeFrozen < 0. )  
            TimeFrozen = TimeFrozen + 1.f;           // wrap-around  
    }  
    else  
    {  
        glutIdleFunc(Animate);  
        int ms = glutGet(GLUT_ELAPSED_TIME);  
        ms %= MS_PER_CYCLE;           // the value of ms is between 0 and MS_PER_CYCLE-1  
        Time = (float)ms / (float)MS_PER_CYCLE; // makes the value of Time [0.,1.)  
        TimeUnfrozen = Time;  
        TimeElapsed = TimeUnfrozen - TimeFrozen;  
        if( TimeElapsed < 0. )  
            TimeElapsed = TimeElapsed + 1.f;           // wrap-around  
    }  
    break;
```

mjb – November 5, 2023

3

When Drawing

4

```
float time = Time - TimeElapsed;  
if( time < 0. )  
    time = time + 1.f;           // wrap-around
```

When drawing, now use **time** in the same way you used *Time* before. For example:

```
glRotatef( 360.f * time, 0., 1., 0. );  
or  
float y = Amplitude * sinf( 2.f * F_PI * time );
```



mjb – November 5, 2023

4

2

Enhancing Computer Graphics Effects by Writing Shaders



Oregon State
University

Mike Bailey

mjb@cs.oregonstate.edu



This work is licensed under a [Creative Commons
Attribution-NonCommercial-NoDerivatives 4.0
International License](#)



Oregon State
University

Computer Graphics

Shaders.pptx

mjb – August 30, 2024

1

How Many Computers do you see in this Photo? One?



Oregon State
University

Computer Graphics

mjb – August 30, 2024

2

No, There Are Two Computers Here

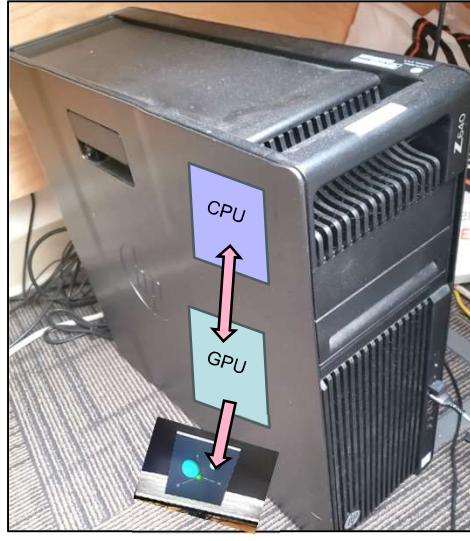
3



Buried within a single chassis, we are tempted to think there is just one computer here.

Oregon State University

Computer Graphics



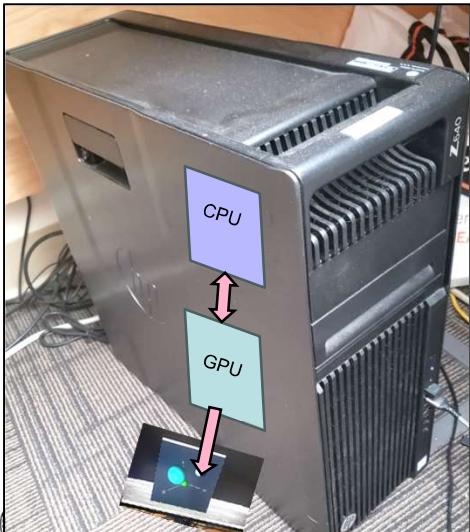
But there are really *two* computers here, a CPU and a GPU. So far, you have been "programming" the GPU by telling OpenGL how to do it for us. This is about to change!

mjb – August 30, 2024

3

No, There Are Two Computers Here

4



We are now going to get into a way-cool part of this class where you get to program the GPU yourself. This is called **Shaders**.

Let's think about it. If you set out to program an external computer, here is what you would need:

1. A programming language
2. A compiler for that language to create an executable
3. A way to see the compiler's error messages
4. A way to download the executable onto the external computer
5. A way to run that executable on the external computer
6. A way to get information into the executable

This sounds like a lot, but it won't turn out to be that big a deal. Trust me!

Oregon State University

Computer Graphics

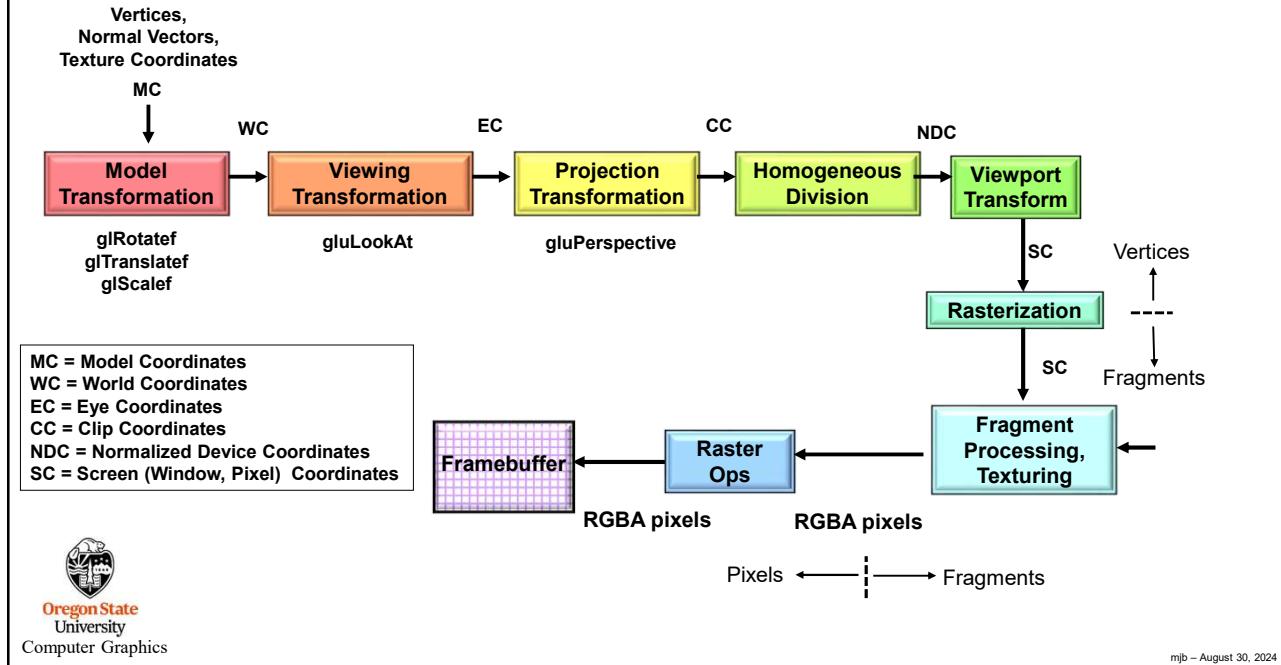
mjb – August 30, 2024

4

2

The Basic Computer Graphics Pipeline, OpenGL-style

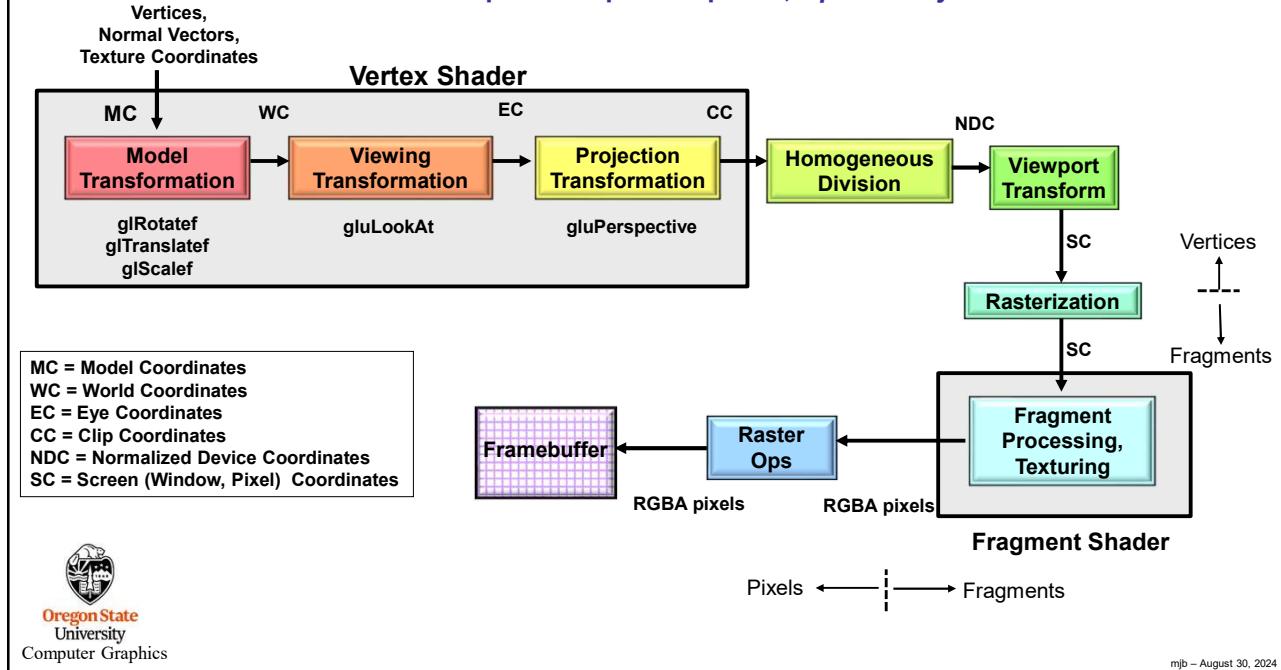
5



5

The Basic Computer Graphics Pipeline, OpenGL-style

6

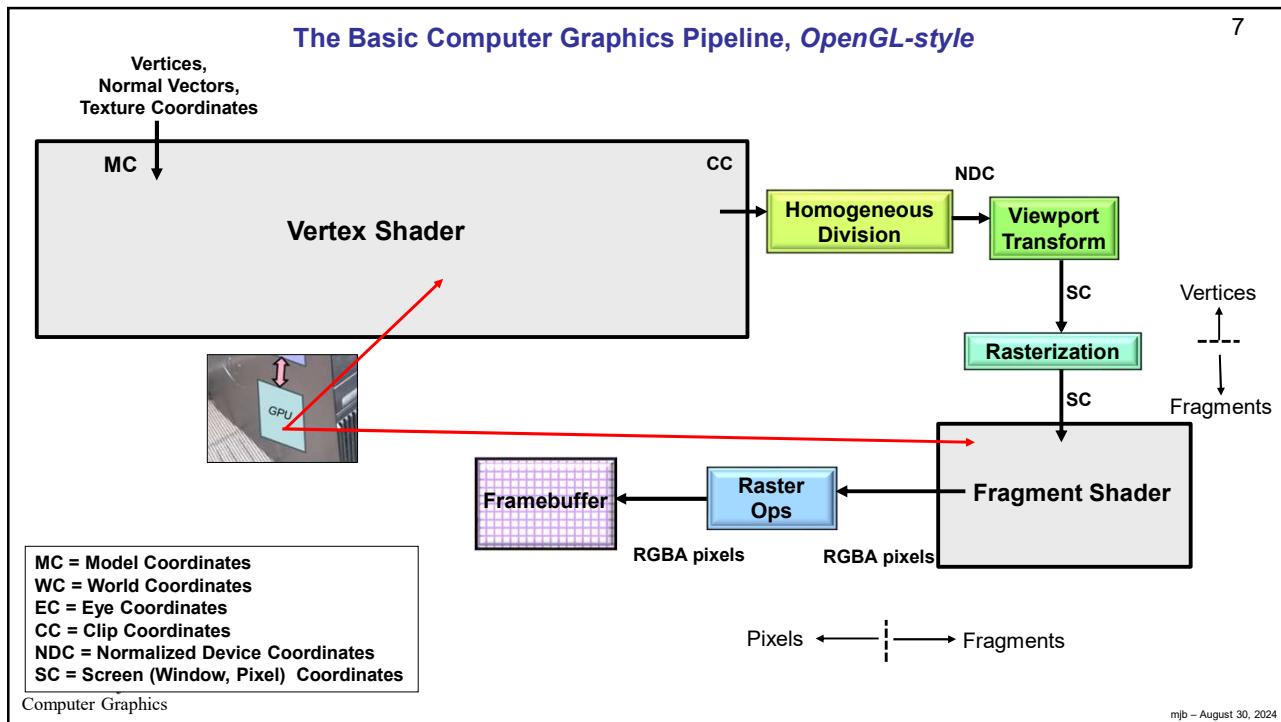


6

3

The Basic Computer Graphics Pipeline, OpenGL-style

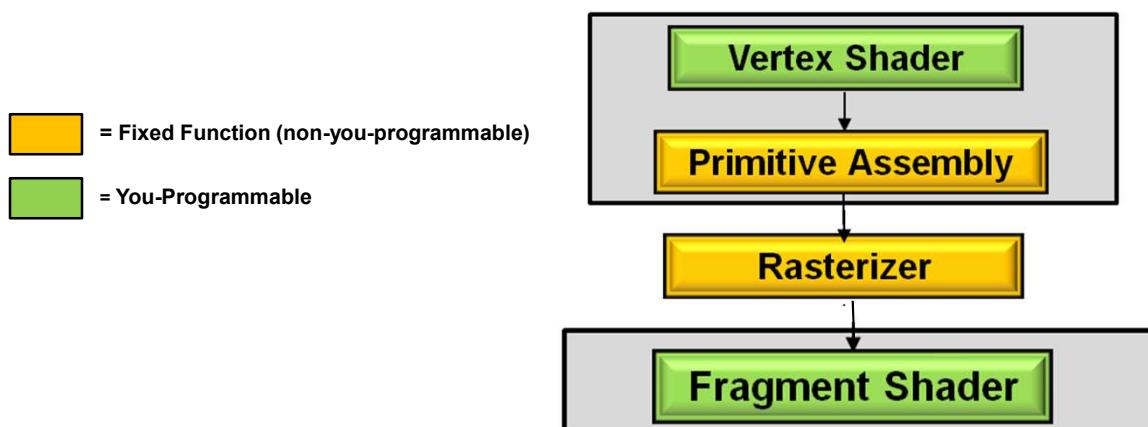
7



7

Our Shaders' View of the Basic Computer Graphics Pipeline

8



There are actually four more GLSL shader types we won't be covering here. In CS 457/557, we will cover all of them.



Computer Graphics

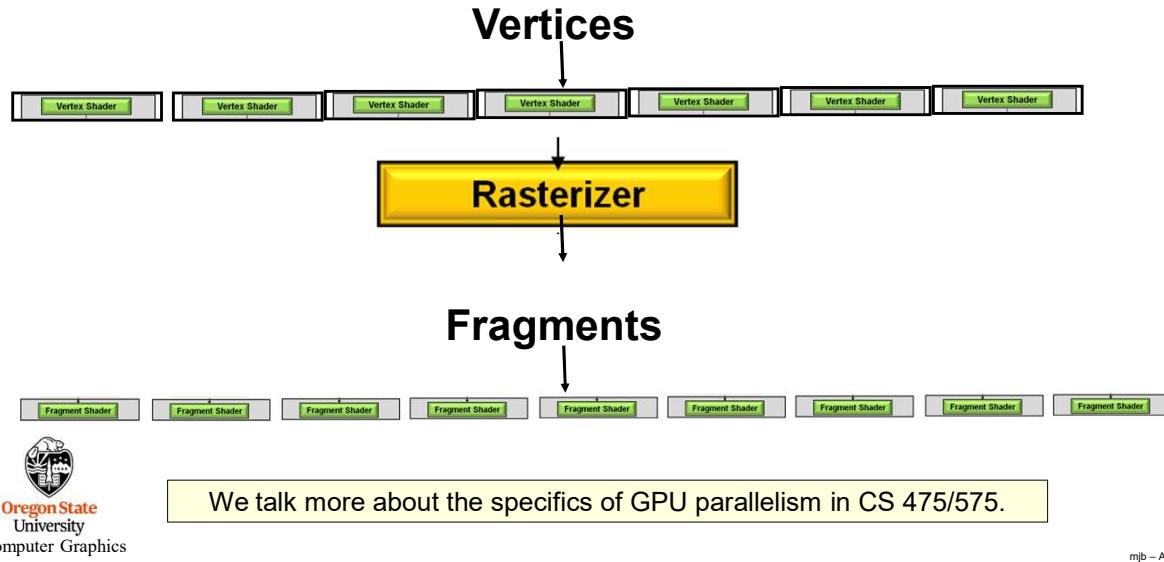
mjb - August 30, 2024

8

4

We Like to Draw the Diagram with One Vertex Shader and One Fragment Shader,
but CG Hardware Achieves Much of its Speed by Handling Hundreds or Thousands of
Vertices and Fragments at the Same Time

9



Oregon State University Computer Graphics

mjb – August 30, 2024

9

A Reminder of what a Rasterizer does

10

There is a piece of hardware called the **Rasterizer**. Its job is to interpolate a line or polygon, defined by vertices, into a collection of **fragments**. Think of it as filling in squares on graph paper.

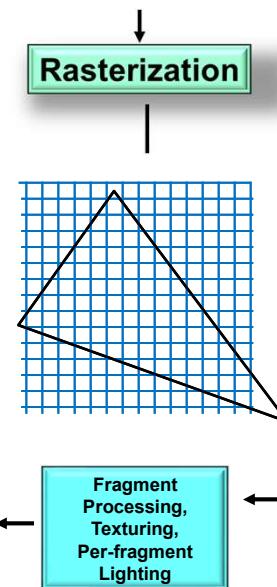
Rasterizers interpolate built-in variables, such as the (x,y) position where the pixel will live and the pixel's z-coordinate. They also interpolate the normal vector (nx, ny, nz) and the texture coordinates (s, t). They can also interpolate user-defined variables as well.

A fragment is a “pixel-to-be”. In computer graphics, “pixel” is defined as having its full RGBA already computed and is headed to be stored in the framebuffer. A fragment does not yet have a computed RGBA, but all of the information needed to compute the RGBA is available.

A fragment is turned into an RGBA pixel by the **fragment processing** operation.

Oregon State University Computer Graphics

mjb – August 30, 2024



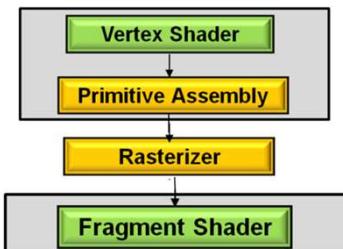
10

5

A GLSL Vertex Shader Takes Over These Operations:

11

- Vertex transformations
- Normal Vector transformations
- Computing per-vertex lighting (although, if you are using shaders anyway, per-fragment lighting looks better)
- Taking per-vertex texture coordinates (s,t) and interpolating them through the rasterizer into the fragment shader

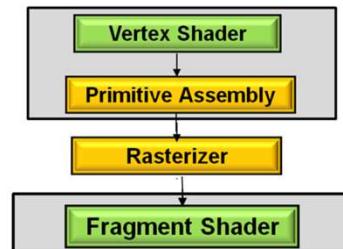


11

A GLSL Fragment Shader Takes Over These Operations:

12

- Color computation
- Texture lookup
- Blending colors with textures (like GL_REPLACE and GL_MODULATE used to do)
- Discarding fragments



12

1. We Need a Programming Language

13

OpenGL developed a shader language called **GLSL: GL Shader Language**.

GLSL is very C-ish, so it should look familiar.



Oregon State
University

Computer Graphics

mjb – August 30, 2024

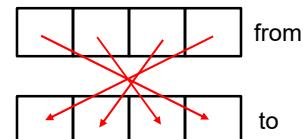
13

GLSL has Many C-Familiar Data Types, plus Extensions for Graphics:

14

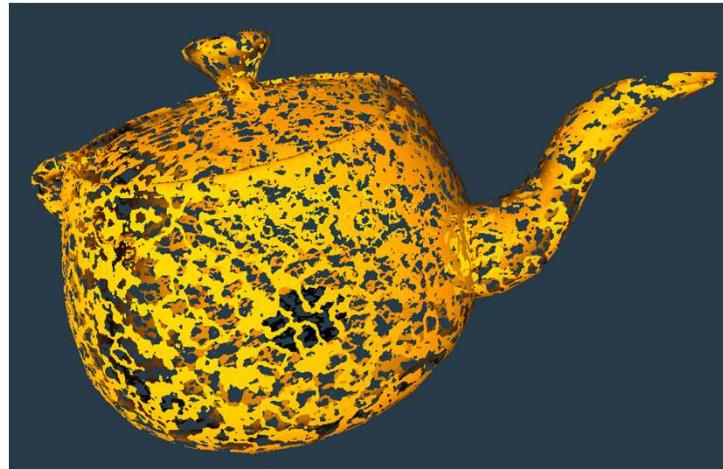
- Types include int, ivec2, ivec3, ivec4
- Types include float, vec2, vec3, vec4
- Types include bool, bvec2, bvec3, bvec4
- Vector components are accessed with .rgba, .xyzw, or.stpq
- Types include mat4
- You can use parallel SIMD operations (doesn't necessarily get implemented in hardware):

```
vec4 a = vec4( 1., 2., 3., 4. );
vec4 b = vec4( 5., 6., 7., 8. );
...
vec4 c = a + b;
```
- Vector components can be "swizzled" (to.abgr = from.rgba)
- Type qualifiers: const, uniform, in, out
- Variables can have "layout qualifiers" to describe how data is stored
- The *discard* operator is used in fragment shaders to get rid of the current fragment



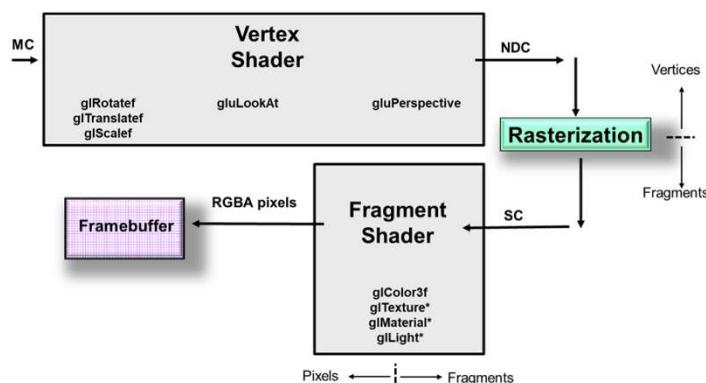
The *discard* Operator Halts Production of the Current Fragment

```
if( random_number < 0.5 )
    discard;
```



GLSL also has Some Different Variable Types to Pass Information Around

- uniform** These are “global” values, assigned into your GLSL program from your C++ program and left alone for a group of primitives. They are read-only accessible from all of your shaders. **They cannot be written to from a shader.**
- out / in** These are passed **out** from the vertex shader stage, interpolated in the rasterizer, and passed **in** to the fragment shader stage.



GLSL has Some *Built-in* Vertex Shader Variables :

17

Input built-ins {

- vec4 gl_Vertex
- vec3 gl_Normal
- vec4 gl_Color
- vec4 gl_MultiTexCoord0
- mat4 gl_ModelViewMatrix
- mat4 gl_ProjectionMatrix
- mat4 gl_ModelViewProjectionMatrix (= gl_ModelViewMatrix * gl_ProjectionMatrix)
- mat3 gl_NormalMatrix (this is the transpose of the inverse of the MV matrix)

Output built-in {

vec4 gl_Position

Note: while this all still works, OpenGL now prefers that you pass in all the above input variables as user-defined *in* variables. We can talk about this later. For now, we are going to use the most straightforward approach possible.



mjb - August 30, 2024

17

GLSL has Some *Built-in* Fragment Shader Variables :

18

Output built-in { vec4 gl_FragColor = the RGBA being sent to the framebuffer



Note: while this all still works, OpenGL now prefers that you pass the RGBA out as a user-defined *out* variable. We can talk about this later. For now, we are going to use the most straightforward approach possible.

mjb - August 30, 2024

18

We haven't forgotten about this.

19

If you set out to program an external computer, here is what you would need:

1. A programming language

GLSL

2. A compiler for that language to create an executable

The GLSL compiler is pre-built into the OpenGL driver. You've already got it.

3. A way to see the compiler's error messages
4. A way to download the executable onto the external computer
5. A way to run that executable on the external computer
6. A way to get information into the executable

We will give you a C++ class to take care of all of this. This is coming up soon.

But, first, let's take a look at what vertex and fragment shader code looks like.



mjb - August 30, 2024

19

My Own Variable Naming Convention

20

With 7 different places that GLSL variables can be created from, I decided to adopt a naming convention to help me recognize what program-defined variables came from what sources:

Beginning letter(s)	Means that the variable ...
a	Is a per-vertex in (attribute) from the application
u	Is a uniform variable from the application
v	Came from the vertex shader
tc	Came from the tessellation control shader
te	Came from the tessellation evaluation shader
g	Came from the geometry shader
f	Came from the fragment shader



This isn't part of "official" GLSL – it is just *my* way of handling the chaos

mjb - August 30, 2024

20

10

The Minimal Vertex and Fragment Shader

21

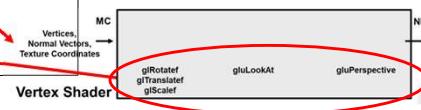
A Vertex Shader is automatically called once per vertex:

```
#version 330 compatibility
```

```
void
main( )
{
    gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
}
```

```
vec4 gl_Vertex
vec3 gl_Normal
vec4 gl_Color
vec4 gl_MultiTexCoord0
mat4 gl_ModelViewMatrix
mat4 gl_ProjectionMatrix
mat4 gl_ModelViewProjectionMatrix (= gl_ModelViewMatrix * gl_ProjectionMatrix)
mat3 gl_NormalMatrix (this is the transpose of the inverse of the MV matrix)

vec4 gl_Position
```



Rasterizer

A Fragment Shader is automatically called once per fragment:

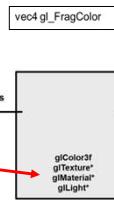
```
#version 330 compatibility
```

```
void
main( )
{
    gl_FragColor = vec4( .5, 1., 0., 1. );
}
```

This code assigns a fixed color ($r=0.5$, $g=1.$, $b=0.$) and alpha ($=1.$) to each fragment drawn

Oregon
University
Computer Graphics

Not terribly useful yet ...



mjb - August 30, 2024

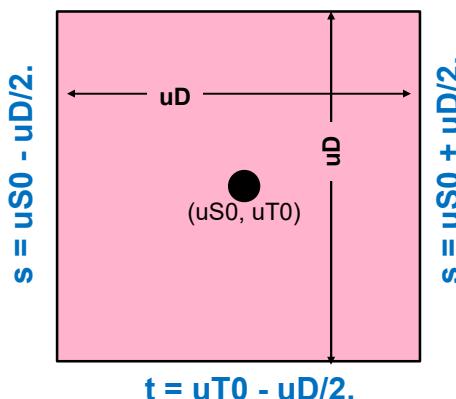
21

A Little More Interesting, I: What if we Want to Color in a Pattern?

22

This pattern example is defined by three uniform variables: $uS0$, $uT0$, and uD , all in texture coordinates (0.-1.). ($uS0, uT0$) are the center of the pattern. uD is the length of each edge of the pattern. The s and t boundaries of the pattern are like this:

$$t = uT0 + uD/2.$$



- $uS0$, $uT0$ are the center of the pattern in (0.-1.) texture coordinates
- uD is the size of the pattern in (0.-1.) texture coordinates

Oregon State
University
Computer Graphics

mjb - August 30, 2024

22

11

A Little More Interesting, II: Getting the Texture Coordinates from the Vertex Shader to the Fragment Shader

23

The vertex shader needs to pass the texture coordinates to the rasterizer so that each fragment shader gets it:

A Vertex Shader is automatically called once per vertex:

```
#version 330 compatibility

out vec2 vST;

void
main( )
{
    vST = gl_MultiTexCoord0.st;           // a vertex's (s,t) texture coordinates
    gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
}
```



The texture coordinates need to come from the *vertex shader* because they were assigned to each *vertex* to begin with

mjb – August 30, 2024

23

A Little More Interesting, III: Drawing a Pattern with the Fragment Shader

24

The fragment shader answers the question: “Am I (the current fragment) inside the pattern or outside it?”

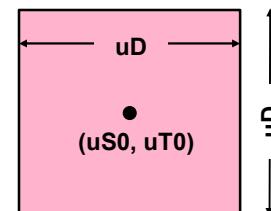
A Fragment Shader is automatically called once per fragment:

```
#version 330 compatibility
uniform float uS0, uT0, uD; // from your program
in vec2 vST;                // from the vertex shader, interpolated through the rasterizer

void
main( )
{
    float s = vST.s;          // the s coordinate of where this fragment is
    float t = vST.t;          // the t coordinate of where this fragment is
    vec3 myColor = vec3( 1., 0.5, 0. );           // default color

    if(      uS0 - uD/2. <= s && s <= uS0 + uD/2. &&
            uT0 - uD/2. <= t && t <= uT0 + uD/2. )
    {
        myColor = vec3( 1., 0., 0. ); // new pattern color
    }

    ...
    gl_FragColor = << myColor with lighting applied >>
}
```



mjb – August 30, 2024

24

A Little More Interesting, IV: Drawing a Pattern with the Fragment Shader

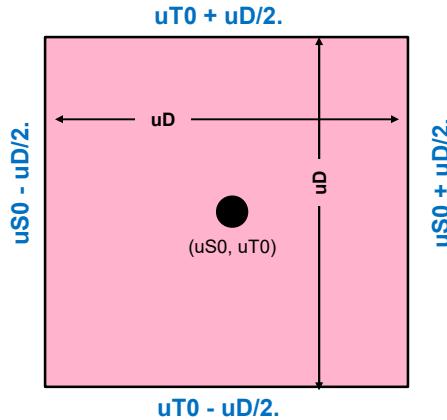
25

The fragment shader answers the question: “Am I (the current fragment) inside the pattern or outside it?”

```

if( uS0 - uD/2. <= s && s <= uS0 + uD/2. && uT0 - uD/2. <= t && t <= uT0 + uD/2. )
{
    myColor = vec3( 1., 0., 0. );
        // change the pattern color if inside the pattern boundaries
}

```



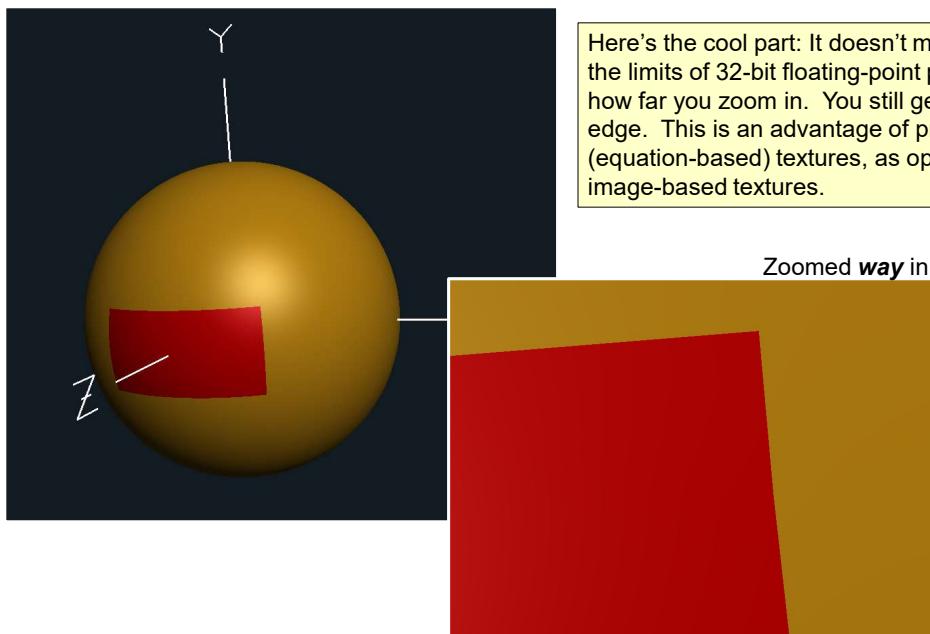
All 4 of these must be true to conclude this fragment is inside the pattern!

- $uS0, uT0$ are the center of the pattern in (0.-1.) texture coordinates
- uD is the size of the pattern in (0.-1.) texture coordinates

Drawing a Pattern on an Object

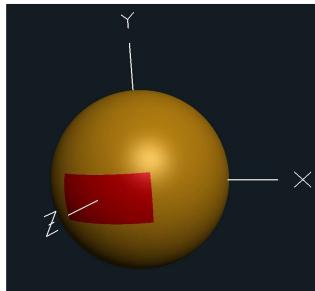
26

Here's the cool part: It doesn't matter (up to the limits of 32-bit floating-point precision) how far you zoom in. You still get a crisp edge. This is an advantage of procedural (equation-based) textures, as opposed to image-based textures.

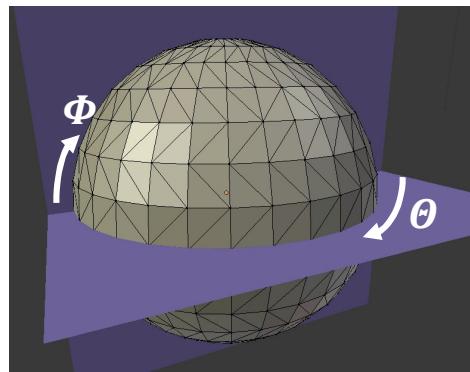


If the Equation Defines a Square, Why Does the Pattern Look Like a Rectangle?

27



$$s = \frac{\Theta - (-\pi)}{2\pi} \quad t = \frac{\Phi - (-\frac{\pi}{2})}{\pi}$$



It's because, **in a sphere**, the s coordinate encompasses twice as much angle ($-180^\circ \rightarrow +180^\circ$) as the t coordinate does ($-90^\circ \rightarrow +90^\circ$). So the same amount of "s" produces twice the distance as the same amount of "t". If you care, you can fix it like this:

```
float s = vST.s;
float t = vST.t;
s = 2.*s;
```



mjb – August 30, 2024

27

Per-Vertex Lighting vs. Per-Fragment Lighting

28

In **per-vertex lighting**, like we have done so far, we apply the lighting equation to the parameters at the vertices and then interpolate the color intensities in the rasterizer. This is what is built-in to standard OpenGL.

In **per-fragment lighting**, we will interpolate the parameters through the rasterizer first and then apply the lighting equation in the fragment shader. To do this, requires shaders.

Lighting Type	Vertex Shader	Rasterizer	Fragment Shader
Per-vertex	Apply lighting model to produce color intensities	Interpolate color intensities	Color the fragments
Per-fragment	Send parameters to rasterizer	Interpolate the parameters	Apply lighting model to color the fragments



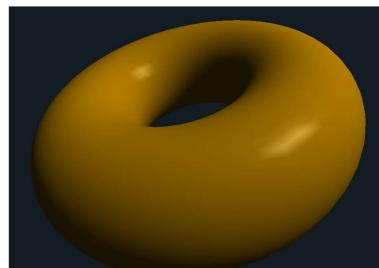
mjb – August 30, 2024

28

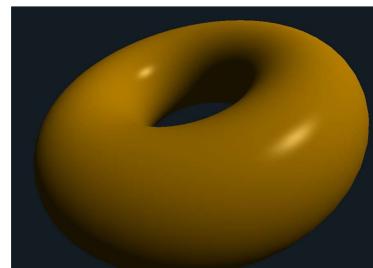
Per-Vertex Lighting vs. Per-Fragment Lighting

29

Per-vertex



Per-fragment



Applying Per-Fragment Lighting, I

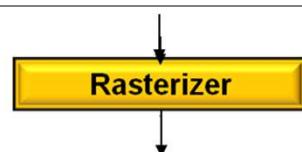
30

Vertex shader:

```
#version 330 compatibility
out vec2 vST;           // texture coords
out vec3 vN;            // normal vector
out vec3 vL;            // vector from point to light
out vec3 vE;            // vector from point to eye

const vec3 LIGHTPOSITION = vec3( 5., 5., 0. );

void
main()
{
    vST = gl_MultiTexCoord0.st;
    vec4 ECposition = gl_ModelViewMatrix * gl_Vertex;           // eye coordinate position
    vN = normalize( gl_NormalMatrix * gl_Normal );               // normal vector
    vL = LIGHTPOSITION - ECposition.xyz;                         // vector from the point to the light position
    vE = vec3( 0., 0., 0. ) - ECposition.xyz;                   // vector from the point to the eye position
    gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
}
```



Applying Per-Fragment Lighting, II

31

Fragment shader:

Here's where we figure out what color this fragment will be, like before

Here's where we apply lighting to that color



Oregon State
University

Computer Graphics

```
#version 330 compatibility
uniform float uKa, uKd, uKs;           // coefficients of each type of lighting
uniform float uShininess;                // specular exponent
in vec2 vST;                           // texture cords
in vec3 vN;                            // normal vector
in vec3 vL;                            // vector from point to light
in vec3 vE;                            // vector from point to eye
void main( )
{
    vec3 Normal = normalize(vN);
    vec3 Light   = normalize(vL);
    vec3 Eye     = normalize(vE);

    vec3 myColor = vec3( 1.0, 0.5, 0.0 );           // default color
    vec3 mySpecularColor = vec3( 1.0, 1.0, 1.0 );    // specular highlight color

    << possibly change myColor >>

    vec3 ambient = uKa * myColor;
    float d = 0.0;
    float s = 0.0;
    if( dot(Normal,Light) > 0.0 )           // only do specular if the light can see the point
    {
        d = dot(Normal,Light);
        vec3 ref = normalize( reflect(-Light, Normal) );           // reflection vector
        s = pow( max( dot(Eye,ref), 0.0 ), uShininess );
    }
    vec3 diffuse = uKd * d * myColor;
    vec3 specular = uKs * s * mySpecularColor;
    gl_FragColor = vec4( ambient + diffuse + specular, 1.0 );
}
```

August 30, 2024

31

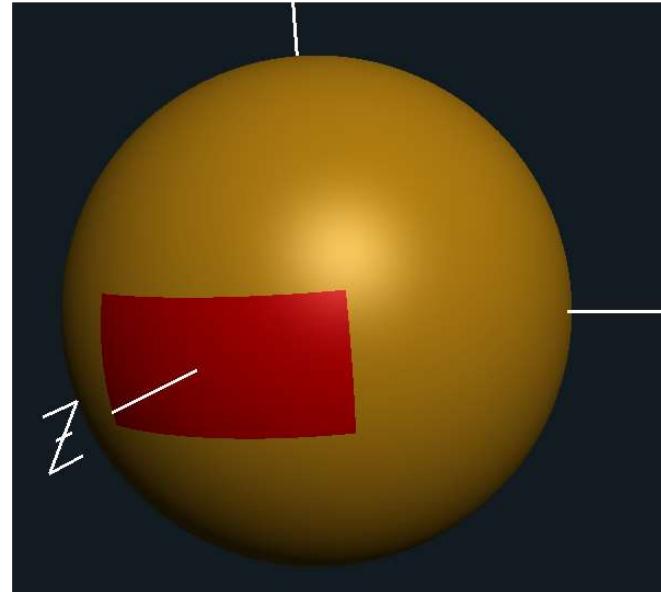
Applying Per-Fragment Lighting, III

32



Oregon State
University

Computer Graphics



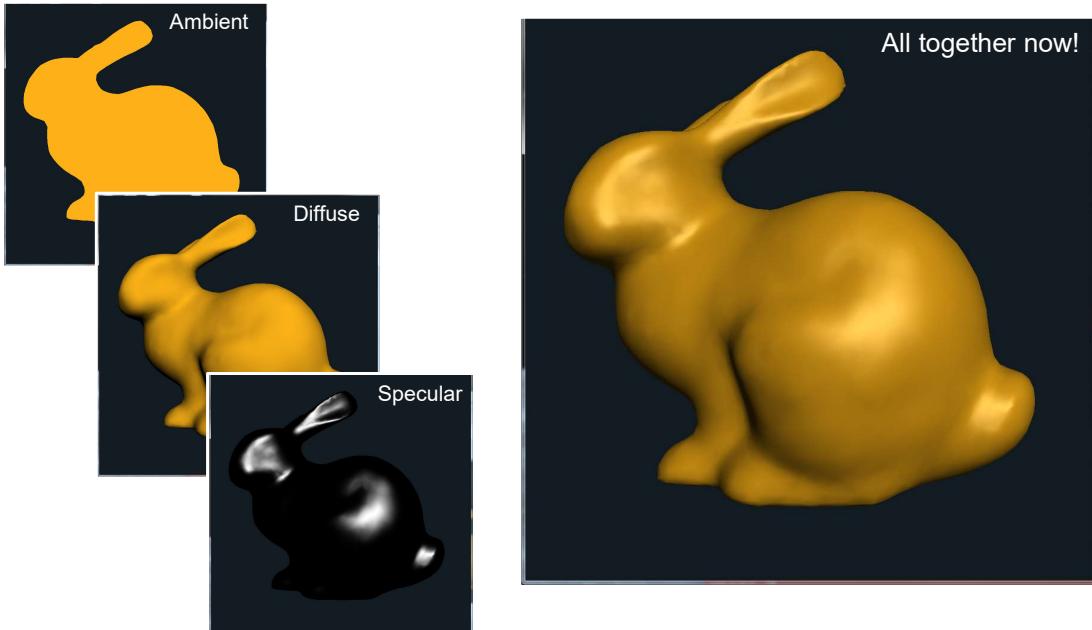
mjb - August 30, 2024

32

16

Per-fragment Lighting is Good, Even Without a Pattern!

33




Oregon State
University
Computer Graphics

mjb – August 30, 2024

33

**Setting up a Shader via the OpenGL API is somewhat Involved:
Here is our C++ Class to Simplify the Shader Setup for You**

34

First, follow these steps:

1. You will see two files that are already in your Sample folder: **glslprogram.h** and **glslprogram.cpp**
2. In your sample.cpp file, un-comment the line:
`#include "glslprogram.cpp"`

These two files have been reduced to have just the shader features you need for Project #6.

If you are not working on Project #6, but are working on something bigger, I have more complete versions of glslprogram.h and glslprogram.cpp – just ask me.


Oregon State
University
Computer Graphics

mjb – August 30, 2024

34

17

**Setting up a Shader via the OpenGL API is somewhat Involved:
Here is our C++ Class to Simplify the Shader Setup for You**

35

Put these in with the Global Variables:

```
GLSLProgram Pattern;           // your VS+FS shader program name  
float      Time;  
#define MS_IN_THE_ANIMATION_CYCLE 10000
```



mjb – August 30, 2024

35

**Setting up a Shader via the OpenGL API is somewhat Involved:
Here is our C++ Class to Simplify the Shader Setup for You**

36

Do this in Animate() like you've always done:

```
void  
Animate( )  
{  
    int ms = glutGet( GLUT_ELAPSED_TIME );                      // milliseconds  
    ms %= MS_IN_THE_ANIMATION_CYCLE;  
  
    Time = (float)ms / (float)MS_IN_THE_ANIMATION_CYCLE;        // [ 0., 1. )  
}
```



mjb – August 30, 2024

36

18

**Setting up a Shader via the OpenGL API is somewhat Involved:
Here is our C++ Class to Simplify the Shader Setup for You**

37

*Do this in InitGraphics() somewhere **after** where the
window has been created and GLEW has been setup:*

In C/C++, the
exclamation
point (!) is
pronounced
"not".

```
Pattern.Init( );
bool valid = Pattern.Create( "pattern.vert", "pattern.frag" );

if( ! valid )
{
    fprintf( stderr, "Yuch! The shader did not compile.\n" );
}
else
{
    fprintf( stderr, "Woo-Hoo! The shader compiled.\n" );
}
```

2. A compiler for that language to create an executable
3. A way to see the compiler's error messages
4. A way to download the executable onto the external computer

This attempts to load, compile, and link the shader program. If something goes wrong,
Pattern.Create() prints error messages into the console window and returns a value of **valid=false**.

37

**Setting up a Shader via the OpenGL API is somewhat Involved:
Here is our C++ Class to Simplify the Shader Setup for You**

38

Do this in Display():

```
float s0 = some function of Time
float t0 = some function of Time
float d  = some function of Time
...
Pattern.Use(); // turns the shader program on
// no more fixed-function – the shader Pattern now handles everything
// but the shader program just sits there idling until you draw something

Pattern.SetUniformVariable( "uS0", s0 );
Pattern.SetUniformVariable( "uT0", t0 );
Pattern.SetUniformVariable( "uD", d );

glCallList( SphereList ); // now the shader program has vertices and fragments to work on

Pattern.UnUse(); // go back to fixed-function OpenGL
```

5. A way to run that executable on the external computer

6. A way to get information into the executable

38

Graphics chips have functionality on them called **Texture Units**. Each Texture Unit is identified by an integer number, typically 0-15, but oftentimes more.

To tell a shader how to get to a specific texture image, assign that texture into a specific **Texture Unit number** and then tell your shader what Texture Unit number to use. Your C/C++ code will look like this:

```
glActiveTexture( GL_TEXTURE5 );           // use texture unit 5
 glBindTexture( GL_TEXTURE_2D, TexName );
```



Computer Graphics

The file gl.h has these lines:
#define GL_TEXTURE0 0x84C0
#define GL_TEXTURE1 0x84C1
#define GL_TEXTURE2 0x84C2
#define GL_TEXTURE3 0x84C3
#define GL_TEXTURE4 0x84C4
#define GL_TEXTURE5 0x84C5
#define GL_TEXTURE6 0x84C6
#define GL_TEXTURE7 0x84C7
#define GL_TEXTURE8 0x84C8
...

mjb - August 30, 2024

The Whole Process Looks Like This, I:

```
// globals:
unsigned char * Texture;
GLuint      TexName;
GLSLProgram Pattern;

...
// In InitGraphics():

 glGenTextures( 1, &TexName );
int nums, numt;
Texture = BmpToTexture( "filename.bmp", &nums, &numt );
glBindTexture( GL_TEXTURE_2D, TexName );
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT );
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT );
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR );
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR );
glTexImage2D( GL_TEXTURE_2D, 0, 3, nums, numt, 0, 3, GL_RGB, GL_UNSIGNED_BYTE, Texture );

Pattern.Init();
bool valid = Pattern.Create( "pattern.vert", "pattern.frag" );
If( !valid )
{
    ...
}
```



Computer Graphics

mjb - August 30, 2024

The Whole Process Looks Like This, II:

41

This is the hardware Texture Unit Number. You can choose anything in the range 0-15.

```
...  
// In Display():  
  
Pattern.Use( );  
glActiveTexture( GL_TEXTURE5 ) // your C++ program specifies that you want the texture to live on texture unit 5  
glBindTexture( GL_TEXTURE_2D, TexName );  
Pattern.SetUniformVariable( "uTexUnit", 5 ) // tell your shader program to find the texture on texture unit 5  
    << draw something >>  
Pattern.UnUse( );
```



Oregon State
University

Computer Graphics

mjb - August 30, 2024

41

2D Texturing within the Shaders

42

Vertex shader:

```
#version 330 compatibility  
out vec2 vST;  
  
void  
main()  
{  
    vST = gl_MultiTexCoord0.st;  
    gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
```

**texture() is a built-in texture map lookup function –
it returns a vec4 (RGBA)**

Rasterizer

Fragment shader:

```
#version 330 compatibility  
in vec2 vST;  
uniform sampler2D uTexUnit;  
  
void  
main()  
{  
    vec3 newcolor = texture(uTexUnit, vST).rgb;  
    gl_FragColor = vec4( newcolor, 1. );
```

Pattern.SetUniformVariable("uTexUnit", 5);

Convert the vec4 rgba from the texture()
call to just the vec3 rgb that we need

Oregon State
University
Computer Graphics

mjb - August 30, 2024

42

21

2D Texturing within the Shaders

43



43

What if You Want to Use Two Textures in a Shader?

44

```
// In Display():
Pattern.Use( );
glActiveTexture( GL_TEXTURE5 );
glBindTexture( GL_TEXTURE_2D, TexName0 );

glActiveTexture( GL_TEXTURE6 );
glBindTexture( GL_TEXTURE_2D, TexName1 );

Pattern.SetUniformVariable("uTexUnit0", 5 );
Pattern.SetUniformVariable("uTexUnit1", 6 );

glCallList( ... );

Pattern.UnUse( );
```

Fragment shader:

```
#version 330 compatibility
in vec2 vST;
uniform sampler2D uTexUnit0;
uniform sampler2D uTexUnit1;

void
main( )
{
    vec3 newcolor0 = texture( uTexUnit0, vST ).rgb;
    vec3 newcolor1 = texture( uTexUnit1, vST ).rgb;
    gl_FragColor = ...
```

44

Why Would You Want to Use More Than One Texture in a Shader?

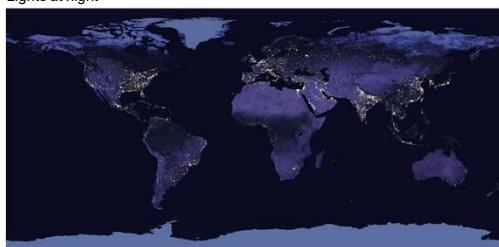
45

Once the RGBs have been read from a texture, they are just numbers. You can do any arithmetic you want with the texture RGBs, other colors, lighting, etc. Here is an example of blending two textures at once:

Daytime



Lights at night



Computer Graphics



mjb – August 30, 2024

45

Why Would You Want to Use More Than One Texture in a Shader?

46

Textures used here:

- Day
- Night
- Heights (bump-mapping)
- Clouds
- Specular highlights



Visualization by Nick Gebbie

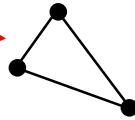
46

Something Goofy: Turning XYZs into RGBs in Model Coordinates

47

Vertex shader:

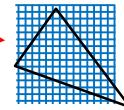
```
#version 330 compatibility
out vec3 vColor; per-vertex
void
main( )
{
    vec4 pos = gl_Vertex;
    vColor = pos.xyz; // set rgb from xyz!
    gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
}
```



Rasterizer

Fragment shader:

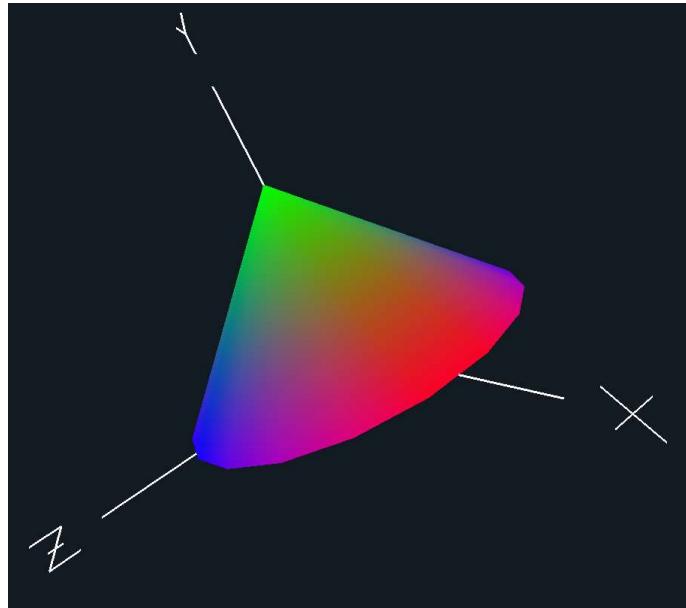
```
#version 330 compatibility
in vec3 vColor; per-fragment
void
main( )
{
    gl_FragColor = vec4( vColor, 1. );
}
```



Setting rgb from the Untransformed xyz, I

48

vColor = gl_Vertex.xyz;



Turning XYZs into RGBs in Eye (World) Coordinates

49

Vertex shader:

```
#version 330 compatibility
out vec3 vColor;

void
main( )
{
    vec4 pos = gl_ModelViewMatrix * gl_Vertex;
    vColor = pos.xyz; // set rgb from xyz!
    gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
}
```

```
#version 330 compatibility
out vec3 vColor;

void
main( )
{
    vec4 pos = gl_Vertex;
    vColor = pos.xyz; // set rgb from xyz!
    gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
}
```

Rasterizer

Fragment shader:

```
#version 330 compatibility
in vec3 vColor;

void
main( )
{
    gl_FragColor = vec4( vColor, 1. );
}
```



mjb – August 30, 2024

49

What's Different About These Two?

50

Set the color from the **untransformed (MC) xyz**

```
#version 330 compatibility
out vec3 vColor;

void
main( )
{
    vec4 pos = gl_Vertex;
    vColor = pos.xyz; // set rgb from xyz!
    gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
}
```

Set the color from the **transformed (WC/EC) xyz**:

```
#version 330 compatibility
out vec3 vColor;

void
main( )
{
    vec4 pos = gl_ModelViewMatrix * gl_Vertex;
    vColor = pos.xyz; // set rgb from xyz!
    gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
}
```



mjb – August 30, 2024

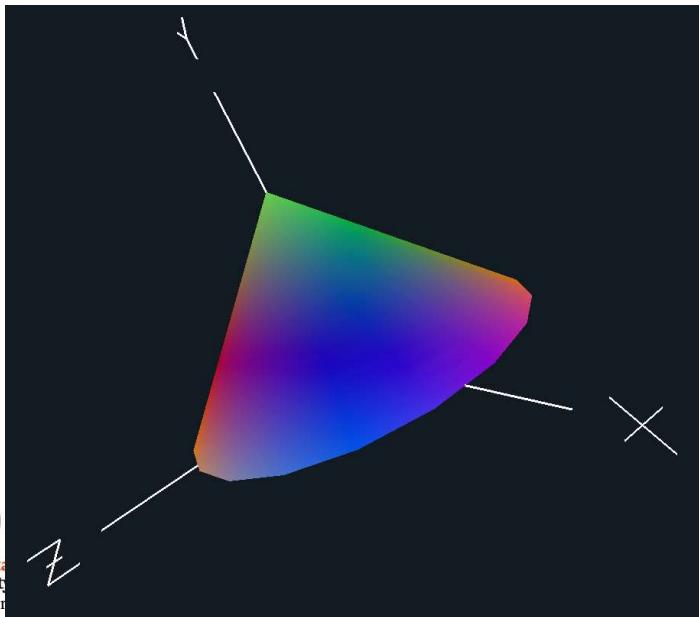
50

25

Setting `rgb` from the Transformed `xyz`, II

51

```
vColor = ( gl_ModelViewMatrix * gl_Vertex ).xyz;
```



Note: the phrase “.xyz” and the phrase “.rgb” mean exactly the same thing: “give me the first 3 numbers from this vec variable”.

What you can't do is mix them, such as “.xgz”



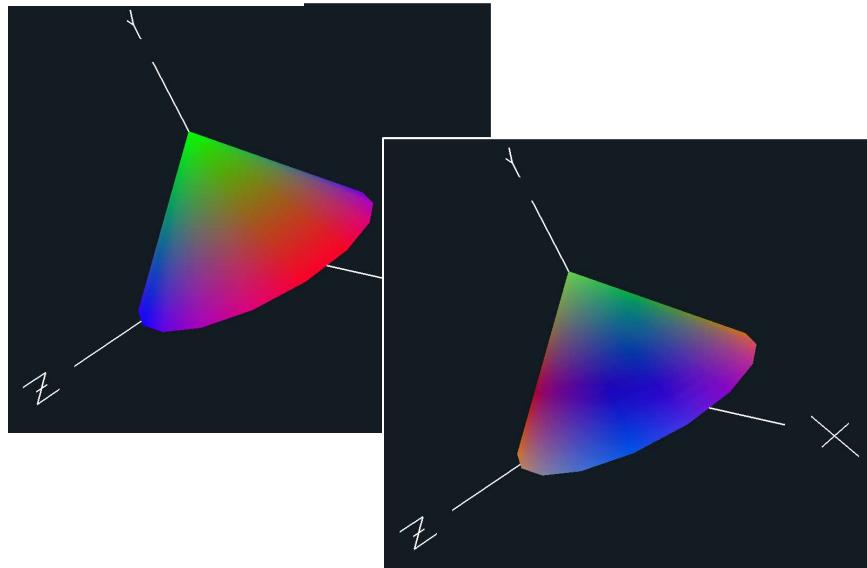
mjb - August 30, 2024

51

Setting `rgb` From `xyz`

52

```
vColor = gl_Vertex.xyz;
```



```
vColor = ( gl_ModelViewMatrix * gl_Vertex ).xyz;
```



mjb - August 30, 2024

52

26

- You need a graphics system that is OpenGL 2.0 or later. Basically, if you got your graphics system in the last 5 years, you should be OK, unless it came from Apple. In that case, who knows how much OpenGL support it has? (The most recent OpenGL level is 4.6)
- Update your graphics driver to the most recent version!
- Do the GLEW setup if you are on Windows. It looks like this in the sample code:

```
GLenum err = glewInit( );
if( err != GLEW_OK )
{
    fprintf( stderr, "glewInit Error\n" );
}
else
    fprintf( stderr, "GLEW initialized OK\n" );
```

This must come **after you've created a graphics window**. (It is this way in the sample code, but I'm saying this because I know some of you go in and "simplify" my sample code by deleting everything you don't think you need.)

- You use the GLSL C++ class you've been given **only after a window has been created and GLEW has been setup**. Only then can you initialize your shader program:

```
Pattern.Init( );
bool valid = Pattern.Create( "pattern.vert", "pattern.frag" );
```

mjb - August 30, 2024

53

A Common Error to Look Out For

54

Here is a piece of code:

```
#version 330 compatibility
out vec3 vColor;

void
main( )
{
    vec4 pos = gl_Vertex;
    vec3 vColor = pos.xyz; // set rgb from xyz!
    gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
}
```



It looks like our example from earlier in these notes. It compiles OK. It should work, right?

Wrong! By re-declaring vColor in "vec3 vColor = pos.xyz", you are making a *local version* of vColor and writing pos.xyz into that local version, not the out variable! The out version of vColor is never getting written to, and so the vColor in the fragment shader will have no sensible value.

Don't ever re-declare in, out, or uniform variables!

 Trust me, you will do this sometime. It's an easy mistake to make mindlessly. I do it every so often myself.

mjb - August 30, 2024

54

Differences if You are on a Mac



55

Unfortunately, Apple froze their GLSL support at version 1.20 – here is how to adapt to that:

- Your shader version number should be 120 (at the top of the .vert and .frag files):
`#version 120 compatibility`

- Instead of the keywords **in** and **out**, use **varying**

- Your OpenGL includes will need to look like this:

```
#include <OpenGL/gl.h>
#include <OpenGL/glu.h>
```

- You don't need to do anything with GLEW

- Your compile sequence will look like this:

```
g++ -framework OpenGL -framework GLUT sample.cpp -o sample -Wno-deprecated
```



mjb – August 30, 2024

55

Guide to Where to Put Pieces of Your Shader Code, I

56

1. Declare the **GLSLProgram** above the main program (i.e., as a global):

```
GLSLProgram Pattern;
```

2. At the end of **InitGraphics()**, create the **shader program** and setup your shaders:

```
Pattern.Init( );
bool valid = Pattern.Create( "pattern.vert", "pattern.frag" );
if( ! valid ) { . . . }
```

3. Turn on the **shader program** in **Display()**, set **shader uniform variables**, draw the objects, then turn off the **shader program**:

```
Pattern.Use( );
Pattern.SetUniformVariable( ... );
glCallList( SphereList( );
Pattern.UnUse( ); // return to the fixed function pipeline
```

4. When you run your program, be sure to check the console window for shader compilation errors!



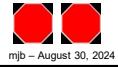
mjb – August 30, 2024

56

28

Tips on drawing the object:

- If you want to key off of s and t coordinates in your shaders, the object must *have* s and t coordinates (vt) assigned to its vertices – *not all OBJ files do!*
- If you want to use surface normals in your shaders, the object must *have* surface normals (vn) assigned to its vertices – *not all OBJ files do!*
- Be sure you explicitly assign *all* of your uniform variables – no error messages occur if you forget to do this – it just quietly screws up.
- The glutSolidTeapot() has been textured in patches, like a quilt – cute, but weird
- The **OsuSphere()** function from the texturing project will give you a very good sphere. Use it, not the GLUT sphere.



mjb – August 30, 2024

1

OpenGL Instancing



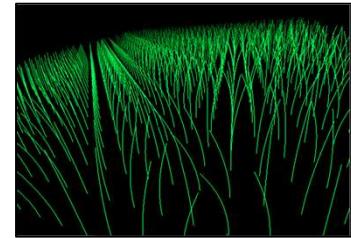
Oregon State
University



This work is licensed under a [Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License](#)

Mike Bailey

mjb@cs.oregonstate.edu



Computer Graphics

Instancing.pptx

mjb – August 27, 2024

1

2

What is Instancing?

Imagine that you needed to draw multiple copies of the same object. Here would be one way to do it (assuming we are using our vertex buffer object C++ class):

```
VertexBufferObject Object;
Object.Init( );

Object.glBegin( GL_LINE_STRIP );
Object glVertex3f( ??, ??, ?? );
...
Object.glEnd( );
...
for( int i = 0; i < numInstances; i++ )
{
    Object.Draw( );
}
```

This would work, but it would require *numInstances* command transmissions from the CPU to the GPU. Is there a better way?



mjb – August 27, 2024

2

1

What is Instancing?

3

OpenGL, like most graphics APIs (Vulkan, for example), supports a concept called **Instancing** in which you specify what to draw and how many times to draw it. Using our C++ class, we would use it like this:

```
VertexBufferObject Object;
Object.Init();

Object.glBegin( GL_LINE_STRIP );
Object glVertex3f( ??, ??, ?? );
...
Object.glEnd();
...
Object.DrawInstanced( numInstances );
```

This only requires *one* command transmission from the CPU to the GPU. It essentially moves the execution of the for-loop over to the GPU side.



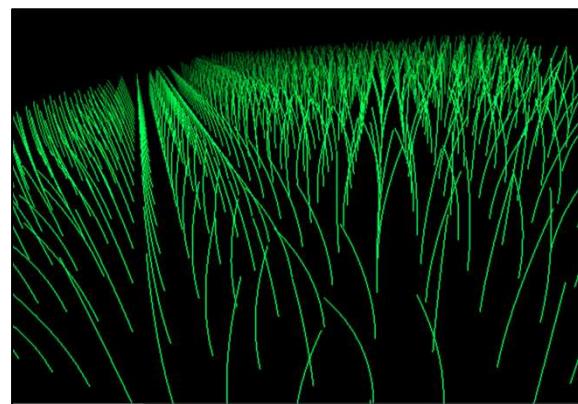
However, the big problem here is that all those copies of the Object *will be drawn identically and on top of each other*. Stay tuned.

mjb – August 27, 2024

3

An Example – Waving Grass

4



We will instance a single blade of grass to make a waving field:



mjb – August 27, 2024

4

Making Each Instance Look Differently

5

There is a built-in vertex shader variable called **gl_InstanceID** that tells us which instance number is being drawn right now. We can use it to change positions, transformations, colors, etc.

Here's how we draw the straight up-and-down blades of grass in a grid:

Vertex shader:

```
#version 330 compatibility
uniform float      uTime;
uniform float      uXmin, uXmax;
uniform float      uYmin, uYmax;
uniform float      uPeriodx, uPeriody;
uniform int        uNumx, uNumy;

const float TWOPI = 2.*3.14159265;

void main( )
{
    int ix = gl_InstanceID % uNumx;
    int iy = gl_InstanceID / uNumx;

    float x = uXmin + float(ix) * (uXmax-uXmin) / float(uNumx-1);
    float y = uYmin + float(iy) * (uYmax-uYmin) / float(uNumy-1);

    vec4 vert = vec4( x, y, gl_Vertex.zw );
    gl_Position = gl_ModelViewProjectionMatrix * vert;
}
```



mjb – August 27, 2024

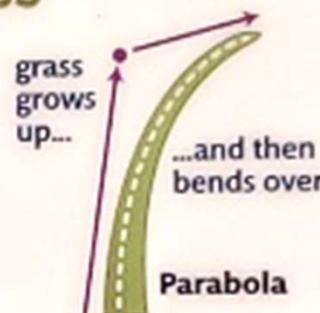
5

An Example – Waving Grass

6

To make the blades sway, we take inspiration from a trick that Pixar used in the movie *A Bug's Life*:

Parabolas as grass
A blade of grass is a small thin curve that can be represented by a parabola.

From the traveling museum exhibit *The Science of Pixar*

mjb – August 27, 2024

6

An Example – Waving Grass

7

The **vertex shader** uses the **gl_InstanceID** built-in variable to place the blades of grass:

```
Vertex shader: void main( )
{
    int ix = gl_InstanceID % uNumx;
    int iy = gl_InstanceID / uNumx;
    float x = uXmin + float(ix) * (uXmax-uXmin) / float(uNumx-1);
    float y = uYmin + float(iy) * (uYmax-uYmin) / float(uNumy-1);

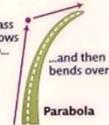
    float kx = cos( TWOPI * uTime * float(ix) / uPeriodx );
    float ky = sin( TWOPI * uTime * float(iy) / uPeriody );

    vec4 vert = vec4( x, y, gl_Vertex.zw );
    float zsq = vert.z*vert.z;
    vert.x += kx * zsq;
    vert.y += ky * zsq;

    gl_Position = gl_ModelViewProjectionMatrix * vert;
}
```

Parabolas as grass

A blade of grass is a small thin curve that can be represented by a parabola.



Waving Grass – Creating the Shader Program and Setting Uniform Variables in InitGraphics():⁸

```
Grass.Init();
bool valid = Grass.Create( "grass.vert", "grass.frag" );
if( ! valid )
    fprintf( stderr, "Shader cannot be created!\n" );
else
    fprintf( stderr, "Shader created.\n" );
Grass.SetVerbose( false );

Grass.Use();
Grass.SetUniformVariable( "uNumx", NUMX );
Grass.SetUniformVariable( "uNumy", NUMY );
Grass.SetUniformVariable( "uXmin", XMIN );
Grass.SetUniformVariable( "uXmax", XMAX );
Grass.SetUniformVariable( "uYmin", YMIN );
Grass.SetUniformVariable( "uYmax", YMAX );
Grass.SetUniformVariable( "uPeriodx", PERIODX );
Grass.SetUniformVariable( "uPeriody", PERIODY );
Grass.UnUse();
```

Waving Grass – Create the Grass-Blade Vertex Buffer Object in InitGraphics():

9

```

Blade.Init();
Blade.glBegin( GL_LINE_STRIP );
for( int i = 0; i < NUMPOINTS; i++ )
{
    float z = ZMIN + (float)i * (ZMAX-ZMIN) / (float)(NUMPOINTS-1);
    Blade glVertex3f( 0., 0., z );
}
Blade glEnd();

```



mjb – August 27, 2024

9

Waving Grass – Drawing the Field of Grass in Display():

10

```

// turn on the shader and set the time:

Grass.Use();
Grass.SetUniformVariable( "uTime", Time );

// draw the grass field:

Blade.DrawInstanced( NUMX*NUMY );

Grass.UnUse();

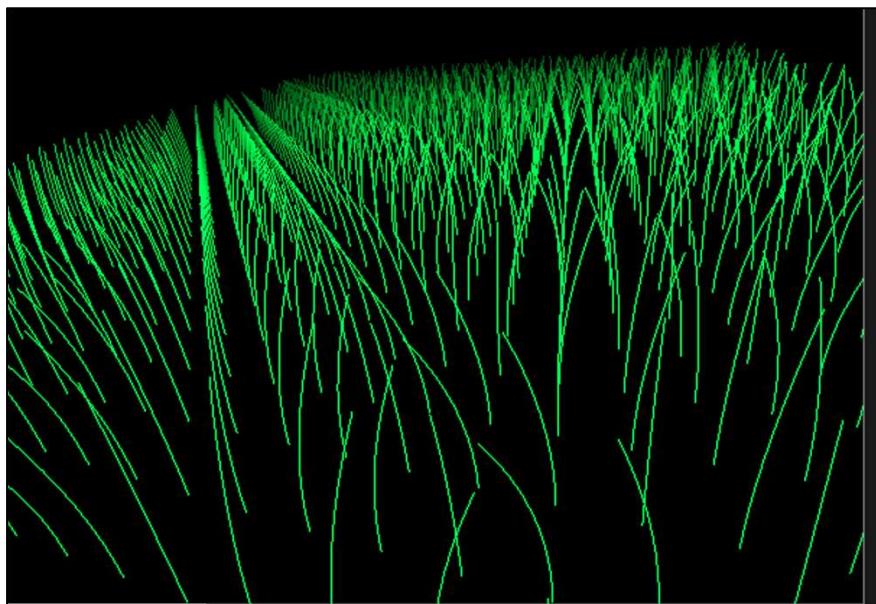
```



mjb – August 27, 2024

10

11



mjb – August 27, 2024

11

OpenGL Picking



Oregon State
University



This work is licensed under a [Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License](#)

Mike Bailey

mjb@cs.oregonstate.edu



Oregon State
University

Computer Graphics

Picking.pptx

mjb – August 22, 2024

What is OpenGL Picking?

In 3D applications, it is useful for the user to point to something and have the program figure out what is being pointed to. This is called **picking**.

You can imagine how you could do this in *software*. You would do all of the transformations yourself and figure out where each object ends up being drawn on the screen and then check its closeness to the cursor. The good news is that this can be done. The bad news is that none of us wants to do it, and it would be painfully slow if we did. Mercifully, OpenGL allows the hardware to help us out. The strategy is straightforward:

1. Tell the hardware that we are in picking (“selection”) mode.
2. Ask the hardware to pretend it is drawing the scene. Go through all the motions, just don’t actually color any pixels.
3. As part of “drawing” the scene, we occasionally give the hardware “names” for the objects we are drawing.
4. Whenever the hardware finds that it would have colored a pixel within a certain tolerance of the cursor, it returns to us the object “name” that it is currently holding. This tells you what set of drawn objects must have passed near the cursor.
5. Tell the hardware that we are back in drawing (“render”) mode.



Oreg
University

Computer Graphics

mjb – August 22, 2024

The Setup

3

Let's look at all the **steps** in detail. The first step is to setup some things in the #defines. This typically includes a picking tolerance (in pixels) and how large to make the array that the hardware will use to record the names that it finds upon a successful pick:

```
// picking tolerance in pixels:  
  
#define PICK_TOL          10.  
  
// how big to make the pick buffer:  
  
#define PICK_BUFFER_SIZE    4096
```

Next, declare the picking buffer and a way to keep track of the rendering mode in the global variables:

```
unsigned int      PickBuffer[ PICK_BUFFER_SIZE ];    // picking buffer  
int              RenderMode = GL_RENDER;           // GL_RENDER or GL_SELECT
```



mjb – August 22, 2024

In InitGraphics()

4

In `InitGraphics()`, tell the hardware what picking array to use and how big it is. You *must* do this *after* you create the window:

```
// open the window and set its title:  
  
glutInitWindowSize( INIT_WINDOW_SIZE, INIT_WINDOW_SIZE );  
glutInitWindowPosition( 0, 0 );  
MainWindow = glutCreateWindow( WINDOWTITLE );  
glutSetWindowTitle( WINDOWTITLE );  
  
...  
  
// setup the picking buffer:  
  
glSelectBuffer( PICK_BUFFER_SIZE, PickBuffer );
```



mjb – August 22, 2024

Assigning the Pick “Names” (numbers)

5

Now comes the part that requires you to plan some strategy: assigning pick names. The names are really unsigned 32-bit integer numbers that get passed down to the hardware during display. The hardware will tell you what name(s) it is holding at the moment a pick occurs. Thus, use enough unique names to establish the identity of the individual objects. For example:

```
glLoadName( 0 );
glutWireSphere( 1.0, 15, 15 );

glLoadName( 1 );
glutWireCube( 1.5 );

glLoadName( 2 );
glutWireCone( 1.0, 1.5, 20, 20 );

glLoadName( 3 );
glutWireTorus( 0.5, 0.75, 20, 20 );
```

would work nicely to distinguish four shapes. Note that you can have as many graphics objects after a call to `glLoadName()` as you'd like. A pick on *any* of those objects, though, will return the same name. You won't know which one caused the pick.

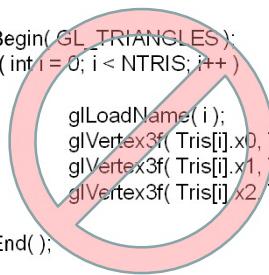
University
Computer Graphics

mjb – August 22, 2024

Pick Names Cannot be Loaded Inside `glBegin`-`glEnd`

6

One thing to be aware of, however, is that a call to `glLoadName()` cannot be embedded inside a `glBegin()` - `glEnd()` pair. Thus, if you are displaying a collection of triangles, you **cannot** say:



```
glBegin(GL_TRIANGLES);
for( int i = 0; i < NTRIS; i++ )
{
    glLoadName( i );
    glVertex3f( Tris[i].x0, Tris[i].y0, Tris[i].z0 );
    glVertex3f( Tris[i].x1, Tris[i].y1, Tris[i].z1 );
    glVertex3f( Tris[i].x2, Tris[i].y2, Tris[i].z2 );
}
glEnd();
```

Instead, you **must** say:

```
for( int i = 0; i < NTRIS; i++ )
{
    glLoadName( i );
    glBegin( GL_TRIANGLES );
        glVertex3f( Tris[i].x0, Tris[i].y0, Tris[i].z0 );
        glVertex3f( Tris[i].x1, Tris[i].y1, Tris[i].z1 );
        glVertex3f( Tris[i].x2, Tris[i].y2, Tris[i].z2 );
    glEnd();
}
```

 Oregon State
University

Computer Graphics

mjb – August 22, 2024

A Better Idea

7

But this is inefficient. So, only do this when you are in selection mode, like this:

```
if( RenderMode == GL_RENDER )
{
    glBegin( GL_TRIANGLES );
    for( int i = 0; i < NTRIS; i++ )
    {
        glVertex3f( Tris[i].x0, Tris[i].y0, Tris[i].z0 );
        glVertex3f( Tris[i].x1, Tris[i].y1, Tris[i].z1 );
        glVertex3f( Tris[i].x2, Tris[i].y2, Tris[i].z2 );
    }
    glEnd();
}
else
{
    for( int i = 0; i < NTRIS; i++ )
    {
        glLoadName( i );
        glBegin( GL_TRIANGLES );
        glVertex3f( Tris[i].x0, Tris[i].y0, Tris[i].z0 );
        glVertex3f( Tris[i].x1, Tris[i].y1, Tris[i].z1 );
        glVertex3f( Tris[i].x2, Tris[i].y2, Tris[i].z2 );
        glEnd();
    }
}
```



Computer Graphics

mjb – August 22, 2024

An Even Better Idea

8

Or, better yet, put each in its own display list:

```
if( RenderMode == GL_RENDER )
{
    glCallList( RenderDisplayList );
}
else
{
    glCallList( PickDisplayList );
}
```



Computer Graphics

mjb – August 22, 2024

The Pick Names Can Be Hierarchical

9

The pick names can be more sophisticated than just a single name. The name-holding place in the hardware is actually a *stack*, so multiple names can be pushed onto the stack. This is useful for hierarchical situations such as:

```
glLoadName( JAGUAR );
glPushName( BODY );
    glCallList( JagBodyList );
glPopName( );

glPushName( FRONT_LEFT_TIRE );
    glPushMatrix();
        glTranslate( ??, ??, ?? );
        glCallList( TireList );
    glPopMatrix();
glPopName( );

glPushName( FRONT_RIGHT_TIRE );
    ...
    ...

glLoadName( PRIUS );
glPushName( BODY );
    glCallList( PriusBodyList );
    ...
```



so that by looking at the 2 pick names that are returned, you know which car *and* what part of that car was picked.

mjb - August 22, 2024

In MouseButton()

10

```
if( ( ActiveButton & LEFT ) && ( status == GLUT_DOWN ) )
{
    RenderMode = GL_SELECT;
    glRenderMode( GL_SELECT );
    Display();
    RenderMode = GL_RENDER;
    NumHits = glRenderMode( GL_RENDER );
    int index = 0;
    for( int i = 0; i < NumHits; i++ )
    {
        unsigned nt numItems = PickBuffer[index++];
        unsigned int zmin = PickBuffer[index++];
        unsigned int zmax = PickBuffer[index++];
        for( int j = 0; j < numItems; j++ )
        {
            unsigned int item = PickBuffer[index++];
            << item is one of your pick names >>
        }
    }
    ActiveButton &= ~LEFT;
    glutSetWindow( GrWindow );
    glutPostRedisplay();
}
if( NumHits == 0 )
{
    // didn't pick anything --
    // use the left mouse for rotation
    ...
}
```



mjb - August 22, 2024

In MouseButton(), Handling the Closest-Hit Option

11

```
int closestItem = -1;
int closestZ = 0xffffffff;

int index = 0;
for (int i = 0; i < NumHits; i++)
{
    unsigned int numItems = PickBuffer[index++];
    unsigned int zmin = PickBuffer[index++];
    unsigned int zmax = PickBuffer[index++];
    for (int j = 0; j < numItems; j++)
    {
        int item = PickBuffer[index++];
        SomethingPicked = true;
        if (ClosestOnOff == OFF)
        {
            Picked[item] = ! Picked[item];
        }
        else
        {
            if (zmin < closestZ)
            {
                closestZ = zmin;
                closestItem = item;
            }
        }
    }
}
if (ClosestOnOff == ON && closestItem >= 0)
{
    Picked[closestItem] = ! Picked[closestItem];
    NumHits = 1;
}
```



mjb – August 22, 2024

What is Left in the Pick Buffer?

12

How many names were on the stack when this hit occurred
unsigned integer representation to tell which of the objects picked is closest to you. A low value of zmin or zmax is closer to you than a high value.

numItems names will be listed here

numItems
zmin
zmax
List of names on the name stack when the pick happened
numItems
zmin
zmax
List of names on the name stack when the pick happened

One of these per pick hit – NumHits total of these



mjb – August 22, 2024

In Display(), I

13

```
...
int vx = glutGet( GLUT_WINDOW_WIDTH );
int vy = glutGet( GLUT_WINDOW_HEIGHT );
...
glMatrixMode( GL_PROJECTION );
glLoadIdentity();
if( RenderMode == GL_SELECT )
{
    int viewport[4];           // place to retrieve the viewport numbers
    glGetIntegerv( GL_VIEWPORT, viewport );
    gluPickMatrix( (double)Xmouse, (double)(vy - Ymouse), PICK_TOL, PICK_TOL, viewport );
}
<< the call to glOrtho( ) or gluPerspective( ) goes here >

...
// init the picking buffer:
if( RenderMode == GL_SELECT )
{
    glInitNames( );
    glPushName( 0xffffffff );           // something you will recognize
}

// draw the objects:
<< your graphics drawing and pick name calls go here >
```



Oregon State
University
Computer Graphics

mjb – August 22, 2024

In Display(), II

14

```
<< your graphics drawing and pick name calls go here >

// possibly draw the axes:
if( AxesOnOff == ON && RenderMode == GL_RENDER )
    glCallList( AxesList );           // don't draw in selection mode

// swap the double-buffered framebuffers:
if( RenderMode == GL_RENDER )
    glutSwapBuffers( );           // don't swap buffers in selection mode
```

Note:

1. When picking, don't draw the axes unless you want to be able to pick them
2. When picking, don't swap the double-buffers



Oregon State
University
Computer Graphics

mjb – August 22, 2024

The only tricky thing is this part:

```
glGetIntegerv( GL_VIEWPORT, viewport);
gluPickMatrix( (double)Xmouse, (double)(vy - Ymouse), PICK_TOL, PICK_TOL, viewport );
```

This code looks at the location and size of your viewport, where the mouse is, and what picking tolerance you wanted and changes your projection matrix so that this pick box region occupies the full window. Thus, the hardware is going to check to see if *anything* gets through the clipping test and thus gets drawn. If it does, then a pick occurred.

Note that the gluPickMatrix() line is written *first* in the program's projection transformations because we want this transformation to happen *last* after all other projection transformations are done.


mjb – August 22, 2024

Picking Tricks, I

16

1. Don't waste picking time looking at things you didn't want to pick anyway. For example, if the axes are not meant to ever be picked, do something like this:

```
if( AxesOnOff == ON && RenderMode == GL_RENDER )
    glCallList( AxesList );
```

2. Don't try to pick raster characters. It doesn't work! I don't know if this is by design or by mistake, but allowing raster characters to be drawn *at all* during a Pick Display will cause the picking to think those characters were picked regardless of where you point the cursor. If you have to draw raster characters, do this:

```
if( RenderMode == GL_RENDER )
{
    glDisable( GL_DEPTH_TEST );
    glMatrixMode( GL_PROJECTION );
    glLoadIdentity();
    gluOrtho2D( 0., 100., 0., 100. );
    glMatrixMode( GL_MODELVIEW );
    glLoadIdentity();
    glColor3f( 1., 1., 1. );
    sprintf( str, " NumHits = %d", NumHits );
    DoRasterString( 1., 1., 0., str );
```


mjb – August 22, 2024

Picking Tricks, II

17

3. Recognizing that, since the user will never see what is being drawn during a Pick Display, you can draw your scene differently than what the user *thinks* he/she is seeing.

For example, a wireframe object normally cannot be picked in the empty space between the lines. But, if you draw a *solid* version of the same thing during the Pick Display, then it will look like a user's pick in the empty space will get the wireframe object, even though *you* know it shouldn't. So, in Display() you could say:

```
if( RenderMode == GL_SELECT )
    glCallList( SolidTorusList );
else
    glCallList( WireTorusList );
```

4. It is an interesting user interface-ism to experiment with the PICK_TOL value. Too small, and you have to be very exact to generate a hit. Too large, and you are likely to pick other things in addition to what you wanted to pick.



Oregon State
University

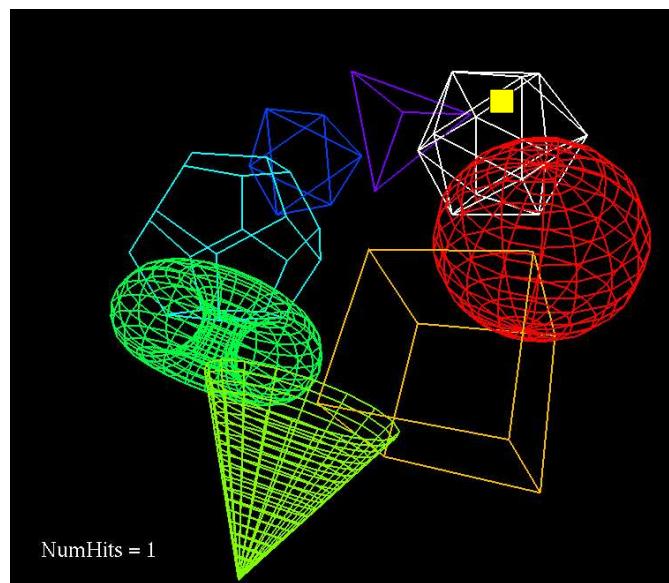
Computer Graphics

mjb – August 22, 2024

Examples, I

18

A pick occurred because an object's pixels went through the pickbox



Oregon State
University

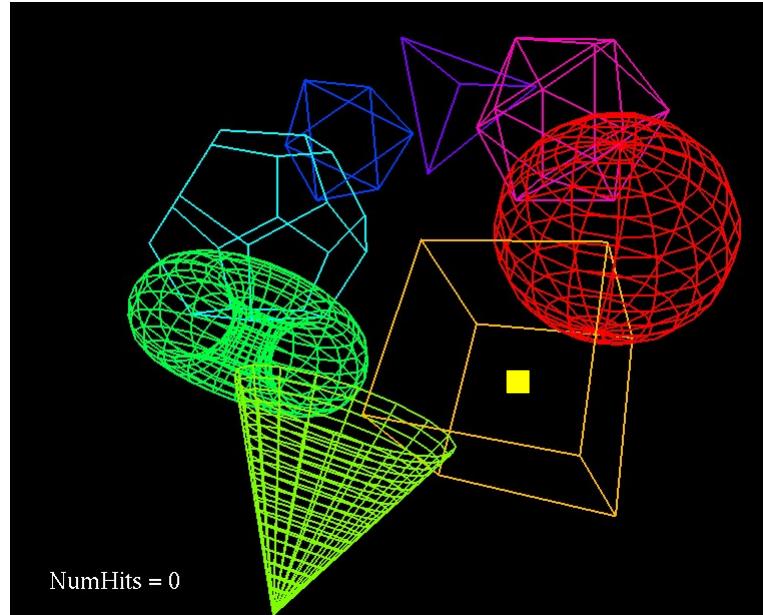
Computer Graphics

mjb – August 22, 2024

Examples, II

19

No pick occurred because no objects' pixels went through the pickbox

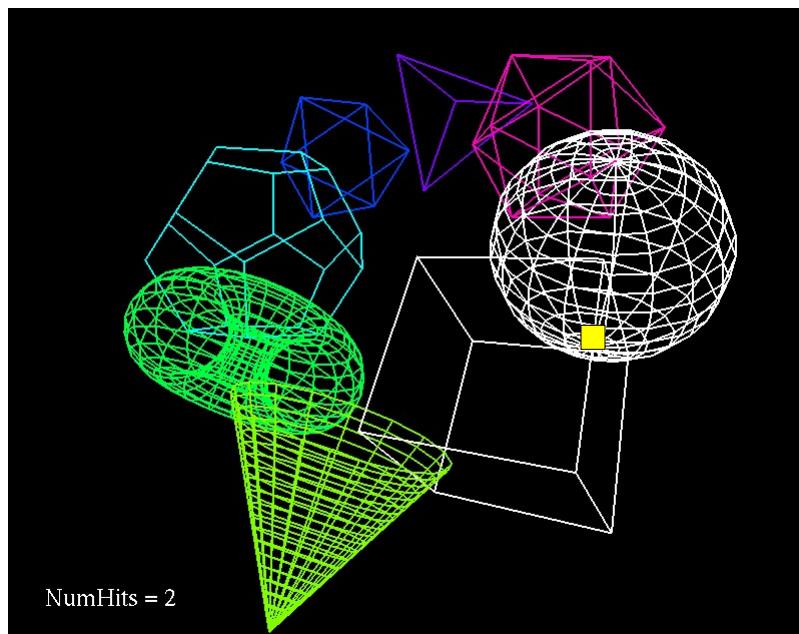


mjb – August 22, 2024

Examples, III

20

Two picks occurred because two object's pixels went through the pickbox



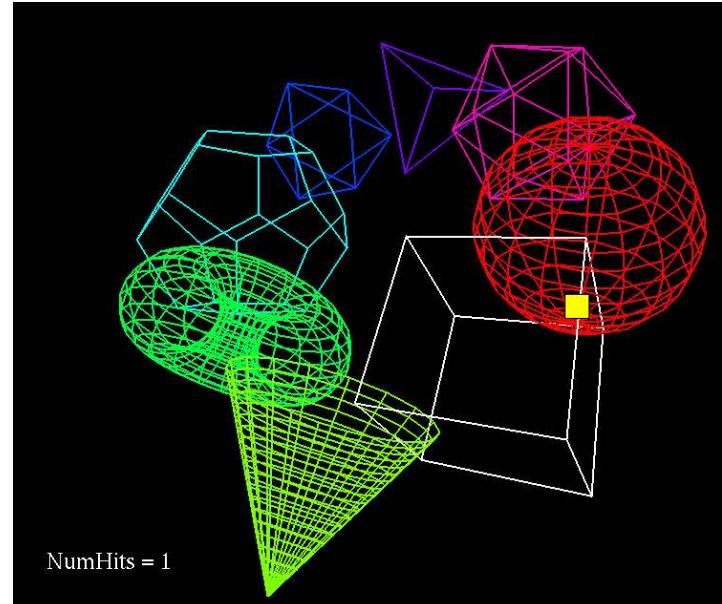
mjb – August 22, 2024

Examples, IV

21



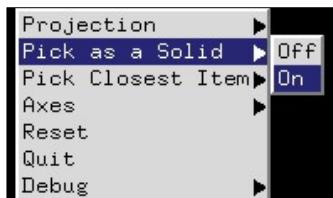
One pick occurred because it was selecting the closest object



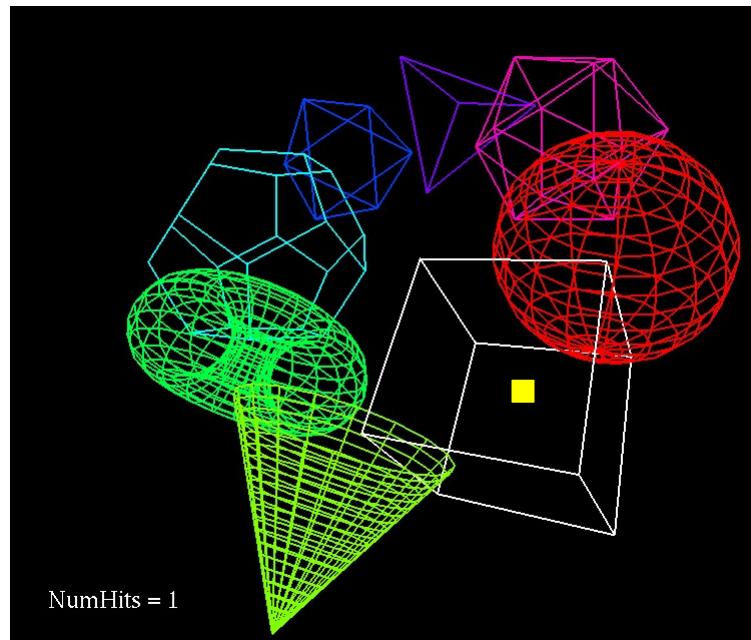
mjb - August 22, 2024

Examples, V

22



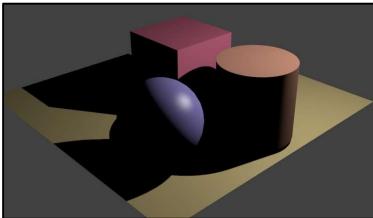
A pick occurred because an object's solid representation pixels went through the pickbox



mjb - August 22, 2024

1

Casting Shadows in OpenGL



This work is licensed under a [Creative Commons
Attribution-NonCommercial-NoDerivatives 4.0
International License](#)



Oregon State
University
Computer Graphics

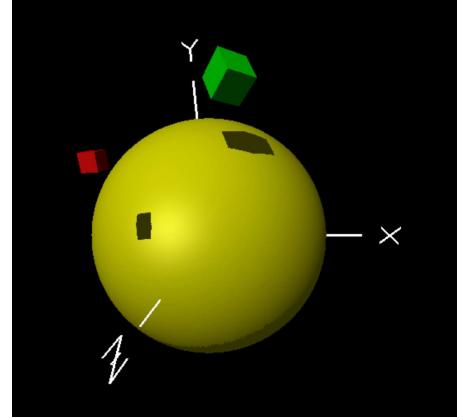


Oregon State

University

Mike Bailey

mjb@cs.oregonstate.edu



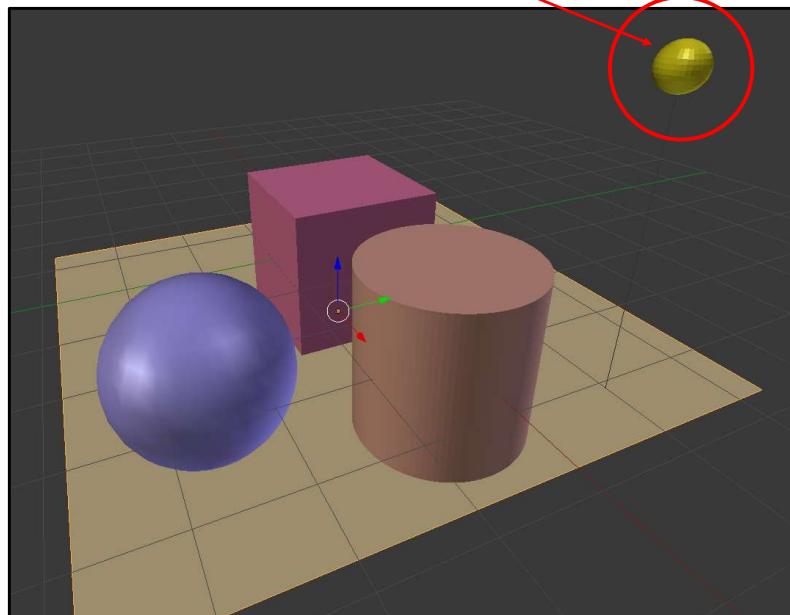
Shadows.pptx

mjb – August 27 2023

1

Identify the Light Source Casting the Shadow

2



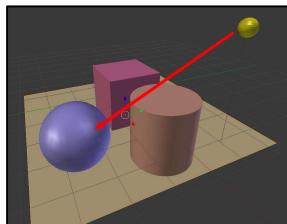
mjb – August 27 2023

2

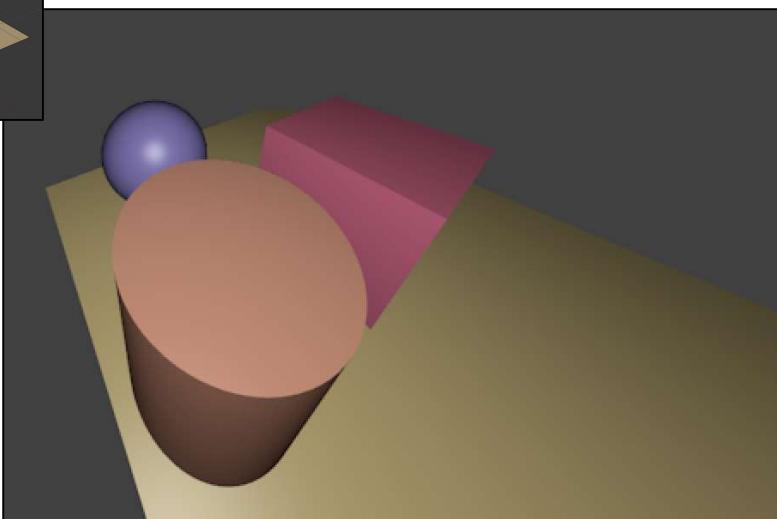
1

First, Render the Scene from the Point of View of that Light Source

3



1. Render a view from the light source – everything you cannot see must be in a shadow

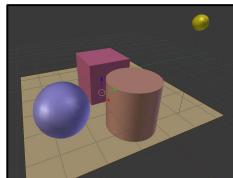


mjb - August 27 2023

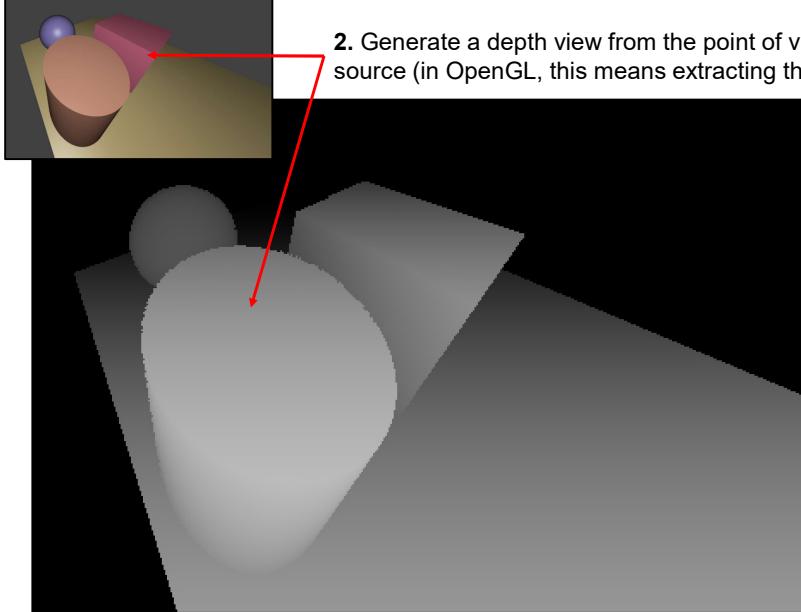
3

Second, use the Z-buffer as a Depth Shadow Map

4



2. Generate a depth view from the point of view of the light source (in OpenGL, this means extracting the z-buffer)



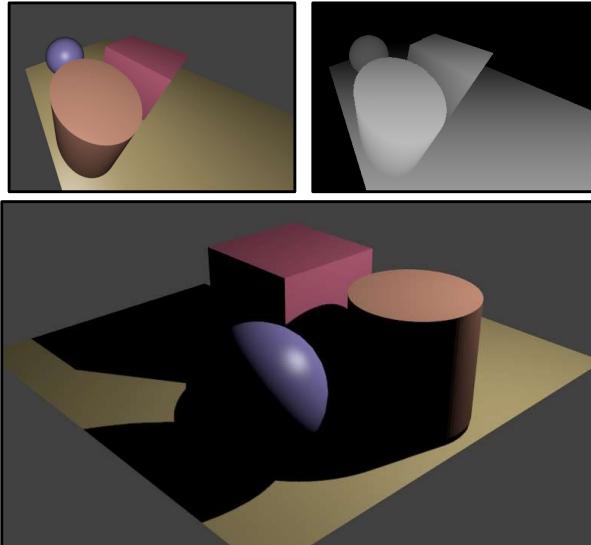
mjb - August 27 2023

4

Third, Render the Scene as you Normally Would, but Consult the Depth Map to Decide where Lighting Can't Reach and Shouldn't Be Used

5

3. Put the eye back where it really belongs. Render that view. Every time you create a pixel in the scene, compare its 3D location against the depth map. If the light-position camera could not see it before, don't allow lighting to be applied to it now.



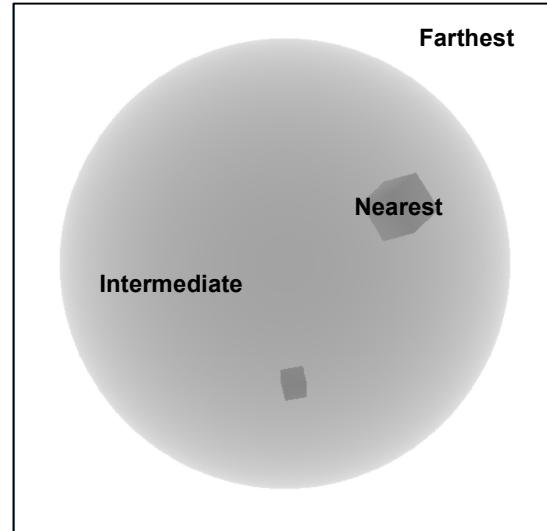
OpenGL Shadow Demo Program: The Depth (Z-buffer) Shadow Map

6

The depth shadow map is created from the point of view of the light source.

The rendering is done into an off-screen framebuffer and only renders the depth (z-buffer), not any colors.

In this grayscale image, dark colors are nearest to the eye, light colors are farther away.



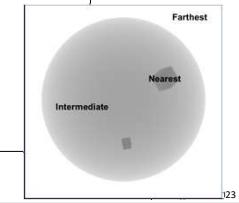
OpenGL Demo Program: Creating the Off-screen Depth Shadow Map Framebuffer

7

```
// create a framebuffer object and a depth texture object:  
glGenFramebuffers(1, &DepthFramebuffer );  
glGenTextures( 1, &DepthTexture );  
  
//Create a texture that will be the framebuffer's depth buffer  
glBindTexture(GL_TEXTURE_2D, DepthTexture );  
glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT, SHADOW_WIDTH, SHADOW_HEIGHT,  
0, GL_DEPTH_COMPONENT, GL_FLOAT, NULL);  
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);  
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);  
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);  
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);  
  
// attach texture to framebuffer as depth buffer:  
glBindFramebuffer(GL_FRAMEBUFFER, DepthFramebuffer);  
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_TEXTURE_2D, DepthTexture, 0);  
  
// force opengl to accept a framebuffer that doesn't have a color buffer in it:  
glDrawBuffer(GL_NONE);  
glReadBuffer(GL_NONE);  
glBindFramebuffer(GL_FRAMEBUFFER, 0);
```

In shadows.cpp:
InitGraphics()

University
Computer Graphics



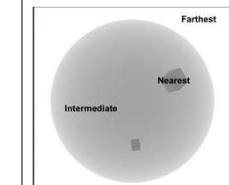
7

OpenGL Demo Program: Rendering into the Depth Shadow Map

8

```
//first pass, render from light's perspective, store depth of scene in texture  
glBindFramebuffer(GL_FRAMEBUFFER, DepthFramebuffer);  
glClear(GL_DEPTH_BUFFER_BIT);  
glDrawBuffer(GL_NONE);  
glReadBuffer(GL_NONE);  
glEnable(GL_DEPTH_TEST);  
glShadeModel(GL_FLAT);  
glDisable(GL_NORMALIZE);  
  
// these matrices are the equivalent of projection and view matrices  
glm::mat4 lightProjection = glm::ortho(-10.0f, 10.0f, -10.0f, 10.0f, 1.f, 20.f);  
glm::vec3 lightPos(LightX, LightY, LightZ);  
glm::mat4 lightView = glm::lookAt(lightPos, glm::vec3(0., 0., 0.), glm::vec3(0., 1., 0.));  
  
//this matrix is the transformation matrix that the vertex shader will use instead of glModelViewProjectionMatrix:  
glm::mat4 lightSpaceMatrix = lightProjection * lightView;  
  
glViewport(0, 0, SHADOW_WIDTH, SHADOW_HEIGHT);  
  
GetDepth.Use( );  
GetDepth.SetUniformVariable((char*)"uLightSpaceMatrix", lightSpaceMatrix);  
glm::vec3 color = glm::vec3(0., 1., 1.);  
GetDepth.SetUniformVariable((char*)"uColor", color);  
DisplayOneScene(GetDepth);  
GetDepth.UnUse( );  
glBindFramebuffer(GL_FRAMEBUFFER, 0);
```

In shadows.cpp:
Display(), l



8

mjb – August 27 2023

OpenGL Demo Program: Rendering using the Depth Shadow Map

9

```

RenderWindow.Use();
RenderWindow.SetUniformVariable((char*)"uShadowMap", 0 );
RenderWindow.SetUniformVariable((char*)"uLightX", LightX);
RenderWindow.SetUniformVariable((char*)"uLightY", LightY);
RenderWindow.SetUniformVariable((char*)"uLightZ", LightZ);
RenderWindow.SetUniformVariable((char*)"uLightSpaceMatrix", lightSpaceMatrix);

glm::vec3 eye = glm::vec3(0., 0., 8.);
glm::vec3 look = glm::vec3(0., 0., 0.);
glm::vec3 up = glm::vec3(0., 1., 0.);
glm::mat4 modelview = glm::lookAt(eye, look, up);

if (Scale < MINSCALE)      Scale = MINSCALE;
glm::vec3 scale = glm::vec3(Scale, Scale, Scale);
modelview = glm::scale(modelview, scale);
glm::vec3 xaxis = glm::vec3(1., 0., 0.);
glm::vec3 yaxis = glm::vec3(0., 1., 0.);
modelview = glm::rotate(modelview, glm::radians(Yrot), yaxis);
modelview = glm::rotate(modelview, glm::radians(Xrot), xaxis);
RenderWindow.SetUniformVariable((char*)"uModelView", modelview);

glm::mat4 proj = glm::perspective(glm::radians(75.f), 1.f, .1f, 100.f);
RenderWindow.SetUniformVariable((char*)"uProj", proj);
DisplayOneScene(RenderWithShadows);
RenderWindow.UnUse( );

```

Computer Graphics

In shadows.cpp:
Display(), II

mjb - August 27 2023

9

OpenGL Shadows Demo Program Shaders: Rendering using the Depth Shadow Map

10

```

#version 330 compatibility

uniform mat4 uLightSpaceMatrix;
uniform mat4 uAnim;

void
main()
{
    gl_Position = uLightSpaceMatrix * uAnim * gl_Vertex;
}

```

GetDepth.vert

```

#version 330 compatibility

uniform vec3 uColor;

void main()
{
    gl_FragColor = vec4(uColor, 1.); // really doesn't matter...
}

```

GetDepth.frag



Oregon State
University
Computer Graphics

mjb - August 27 2023

10

OpenGL Shadows Demo Program Shaders: Rendering using the Depth Shadow Map 11

```
#version 330 compatibility

uniform mat4 uLightSpaceMatrix;
uniform mat4 uAnim;
uniform mat4 uModelView;
uniform mat4 uProj;
uniform float uLightX;
uniform float uLightY;
uniform float uLightZ;

out vec4 vFragPosLightSpace;
out vec3 vNs;
out vec3 vLs;
out vec3 vEs;

void
main()
{
    vec3 LightPosition = vec3(uLightX, uLightY, uLightZ);

    vec4 ECposition = uModelView * uAnim * gl_Vertex;
    vec3 tnorm = normalize( mat3(uAnim) * gl_Normal );
    vNs = tnorm;
    vLs = LightPosition - ECposition.xyz;
    vEs = vec3( 0., 0., 0. ) - ECposition.xyz;

    vFragPosLightSpace = uLightSpaceMatrix * uAnim * gl_Vertex;
    gl_Position = uProj * uModelView * uAnim * gl_Vertex;
}
```

mjb – August 27 2023

11

OpenGL Shadows Demo Program Shaders: Rendering using the Depth Shadow Map 12

```
#version 330 compatibility

uniform vec3 uColor;
uniform sampler2D uShadowMap;
uniform int uShadowsOn;

in vec4 vFragPosLightSpace;
in vec3 vNs;
in vec3 vLs;
in vec3 vEs;

const float BIAS = 0.01;
const vec3 SPECULAR_COLOR = vec3( 1., 1., 1. );
const float SHININESS = 8.;

const float KA = 0.20;
const float KD = 0.60;
const float KS = (1.-KA-KD);

bool IsInShadow(vec4 fragPosLightSpace)
{
    // have to manually do homogenous division to make light space position in range of -1 to 1:
    vec3 projection = fragPosLightSpace.xyz / fragPosLightSpace.w;
    //then make it from 0 to 1:
    projection = 0.5*projection + 0.5;

    //get closest depth from light's perspective
    float closestDepth = texture(uShadowMap, projection.xy).r;

    //get current depth:
    float currentDepth = projection.z;
    bool isInShadow = (currentDepth - BIAS) > closestDepth;
    return isInShadow;
}
```

mjb – August 27 2023

12

OpenGL Shadows Demo Program Shaders: Rendering using the Depth Shadow Map 13

```

void main()
{
    vec3 normal = normalize(vNs);
    vec3 light = normalize(vLs);
    vec3 eye = normalize(vEs);

    float d = 0.;
    float s = 0.;
    vec3 lighting = KA * uColor;

    bool isInShadow = IsInShadow(vFragPosLightSpace);
    if( uShadowsOn != 0 )
        isInShadow = false; // if in ShadowOff mode, nothing should be considered in a shadow
    if( !isInShadow )
    {
        d = dot(normal,light);
        if(d > 0.)
        {
            vec3 diffuse = KD*d*uColor;
            lighting += diffuse;

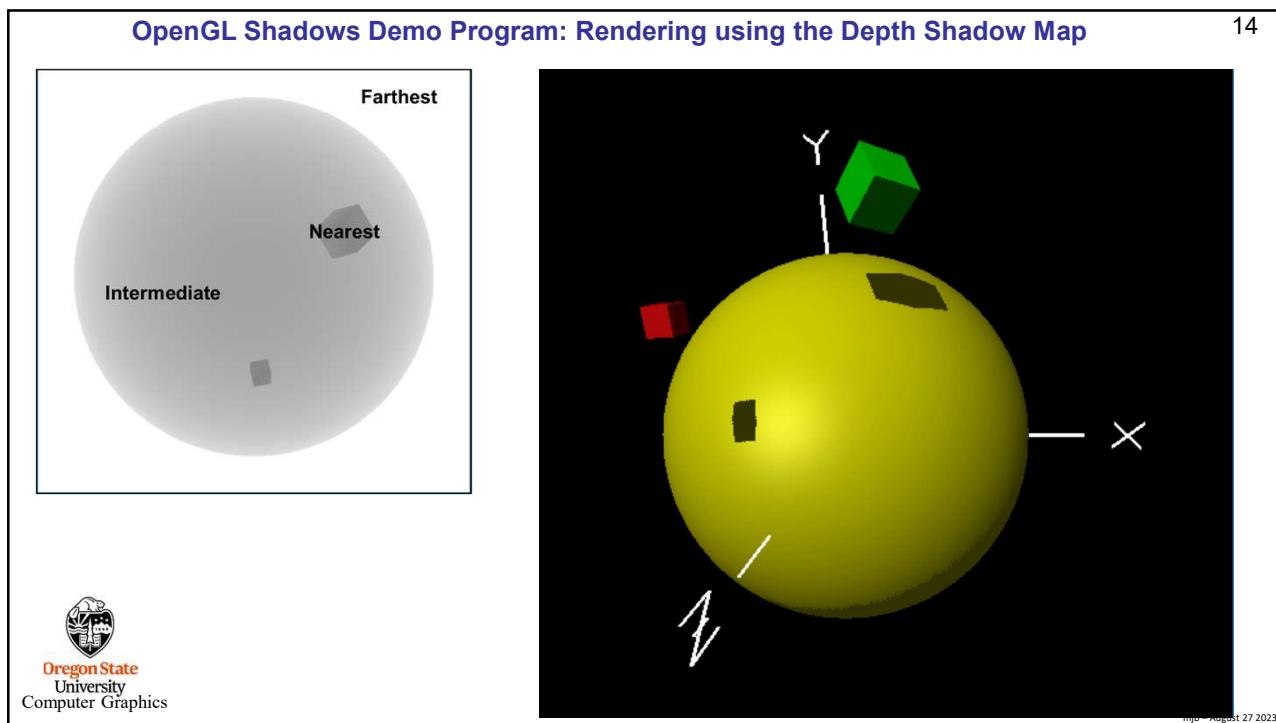
            vec3 refl = normalize( reflect( -light, normal ) );
            float dd = dot(eye,refl);
            if( dd > 0. )
            {
                s = pow( dd, SHININESS );
                vec3 specular = KS*s*SPECULAR_COLOR;
                lighting += specular;
            }
        }
    }
    gl_FragColor = vec4( lighting, 1. );
}

```

Computer Graphics

mjb - August 27 2023

13



14

In `shadows.cpp`**How Did the Demo Program Render that 2D Shadow Map?**

15

```
DisplayShadowMap.Use();
DisplayShadowMap.SetUniformVariable((char*)"uShadowMap", 0 );

glm::mat4 model = glm::mat4(1.f);
DisplayShadowMap.SetUniformVariable((char*)"uModel", model);

glm::vec3 eye = glm::vec3(0., 0., 1.);
glm::vec3 look = glm::vec3(0., 0., 0.);
glm::vec3 up = glm::vec3(0., 1., 0.);
glm::mat4 view = glm::lookAt(eye, look, up);
DisplayShadowMap.SetUniformVariable((char*)"uView", view);

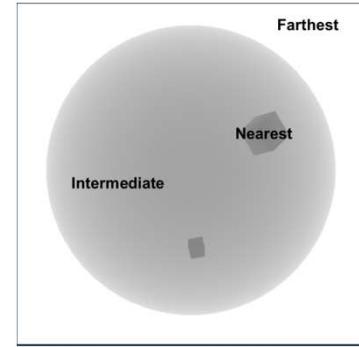
glm::mat4 proj = glm::ortho(-0.6f, 0.6f, -0.6f, 0.6f, .1f, 100.f);
DisplayShadowMap.SetUniformVariable((char*)"uProj", proj);

glBegin(GL_QUADS);
glTexCoord2f(0., 0.);
glVertex3f(-1., -1., 0.);
glTexCoord2f(1., 0.);
glVertex3f( 1., -1., 0.);
glTexCoord2f(1., 1.);
glVertex3f( 1., 1., 0.);
glTexCoord2f(0., 1.);
glVertex3f(-1., 1., 0.);
glEnd();

DisplayShadowMap.UnUse( );
```

University

Computer Graphics



mjb - August 27 2023

15

How Did the Demo Program Render the 2D Shadow Map?

16

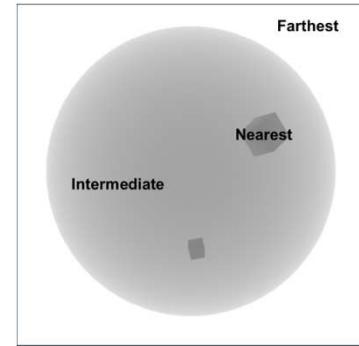
```
#version 330 compatibility

uniform mat4 uModel;
uniform mat4 uView;
uniform mat4 uProj;

out vec2 vST;

void
main()
{
    vST = gl_MultiTexCoord0.st;
    gl_Position = uProj * uView * uModel * gl_Vertex;
}
```

DisplayShadowMap.vert



```
#version 330 compatibility

uniform sampler2D uShadowMap;

in vec2 vST;

void
main( )
{
    float gray = texture(uShadowMap, vST ).r;
    gl_FragColor = vec4( gray, gray, gray, 1. );
}
```

DisplayShadowMap.frag



mjb - August 27 2023

16