Chapter 4

OpenGL 1.1: Light and Material

Dr. Terence van Zyl

University of the Witwatersrand

3rd March 2016

Outline

Light and Material

Introduction

When light strikes a surface, some of it will be reflected. Exactly how it reflects depends in a complicated way on the nature of the surface, what I am calling the material properties of the surface.

Reflection

Definitions

Specular reflection Mirror-like reflection of light rays from a surface. A ray of light is reflected as a ray in the direction that makes the angle of reflection equal to the angle of incidence. A specular reflection can only be seen by a viewer whose position lies on the path of the reflected ray.

Diffuse reflection Reflection of incident light in all directions from a surface, so that diffuse illumination of a surface is visible to all viewers, independent of the viewer's position.

Reflection

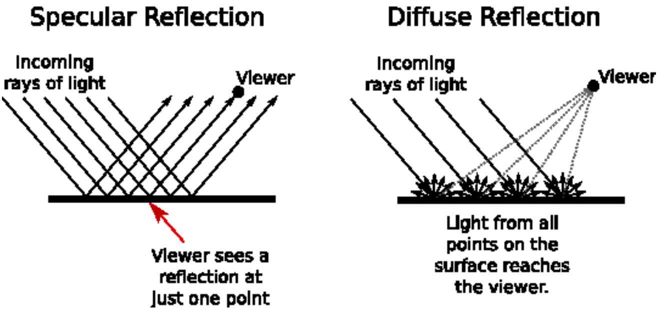


Figure: Specular versus diffuse reflection of light.

Highlights and Shiniiness

Definitions

Specular highlight Illumination of a surface produced by specular reflection. A specular highlight is seen at points on the surface where the angle from the surface to the viewer is approximately equal to the angle from the surface to a light source.

Shininess A material property that determines the size and sharpness of specular highlights. Also called the "specular exponent" because of the way it is used in lighting calculations.

Specular Refecltion Cone

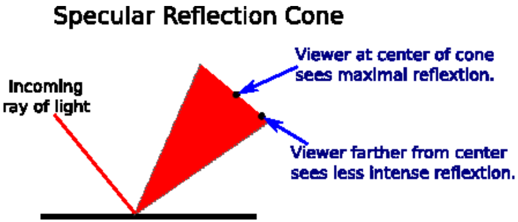


Figure: The specular reflection cone.

Shininess

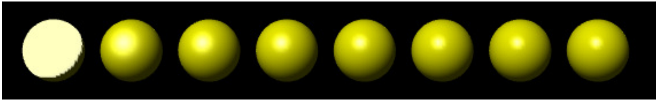


Figure: Form left to right, shininess decreases by 16.

Note

- In OpenGL, shininess is a number in the range 0 to 128.
- As shininess number increases specular highlight decreases.

Light And Colour

Definitions

Diffuse colour A material property that represents the proportion of incident light that is reflected diffusely from a surface.

Specular colour A material property that represents the proportion of incident light that is reflected specularly by a surface.

Lets See It In Action

Demo

Material Properties Demo

Light And Colour

Definitions

Ambient colour A material property that represents the proportion of ambient light in the environment that is reflected by a surface.

Ambient light Directionless light that exists in an environment but does not seem to come from a particular source in the environment. An approximation for light that has been reflected so many times that its original source can't be identified. Ambient light illuminates all objects in a scene equally.

Light And Colour

Definition

Emmision colour A material property that represents colour that is intrinsic to a surface, rather than coming from light from other sources that is reflected by the surface. Emission colour can make the object look like it is glowing, but it does not illuminate other objects. Emission colour is often called "emissive colour."

Alpha

Alpha is only calculated for diffuse light

Light Properties

Introduction

Leaving aside ambient light, the light in an environment comes from a light source such as a lamp or the sun. In fact, a lamp and the sun are examples of two essentially different kinds of light source: a point light and a directional light.

Light Sources

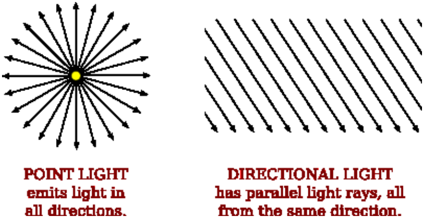


Figure: Point versus directional light.

Light Sources

Definition

Intensity A light source emits light at various wavelengths. The intensity of a light at a given wavelength is the amount of energy in the light at that wavelength. The total intensity of the light is its total energy at all wavelengths. The colour of a light is determined by its intensities at all wavelengths.

Note

- diffuse intensity of a light interacts with diffuse material colour,
- specular intensity of a light interacts with specular material colour,

Normal Vectors

Introduction

The visual effect of a light shining on a surface depends on the properties of the surface and of the light. But it also depends to a great extent on the angle at which the light strikes the surface.

Angles And Vectors Of Surfaces

Definitions

Normal vector A normal vector to a surface at a point on that surface is a vector that is perpendicular to the surface at that point. Normal vectors to curves are defined similarly. Normal vectors are important for lighting calculations.

Unit normal A normal vector of length one; that is, a unit vector that is perpendicular to a curve or surface at a given point on the curve or surface.

Vectors At Vertices

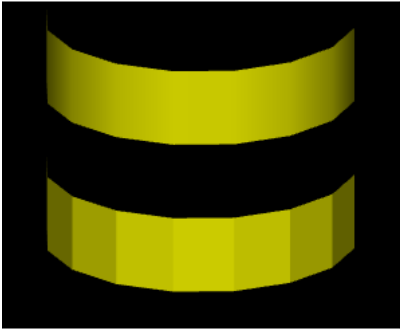


Figure: Using different approximations of vectors perpendicular to the surface.

Vectors At Vertices cont.

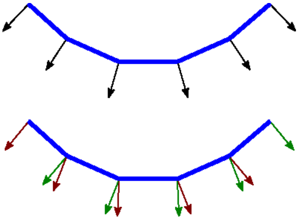


Figure: Using different approximations of vectors perpendicular to the surface.

Lets See It In Action

Demo

Smooth Versus Flat Shading

The OpenGL Lighting Equation

Introduction

The goal of OpenGL lighting calculations is top produce a colour (r, g, b, a) for each point on a surface. These calculations are only done for vertices. For points in between vertices the colour value is interpolated.

The Equation Explained

Assume we have colours of a material:

- ambient $(m_{a,r}, m_{a,g}, m_{a,b})$,
- diffuse $(m_{d,r}, m_{d,g}, m_{d,b})$,
- specular $(m_{s,r}, m_{s,g}, m_{s,b})$, and
- emission $(m_{e,r}, m_{e,g}, m_{e,b})$.

Also assume that the global ambient light is $(l_{a,r}, l_{a,g}, l_{a,b})$. Then the vertex red component is

$$r = m_{e,r} + l_{a,r} * m_{a,r} + l_{0,r} + l_{1,r} + l_{2,r} + \ldots$$

where $l_{i,r}$ is the red contribution from light i.

The Equation Explained Further

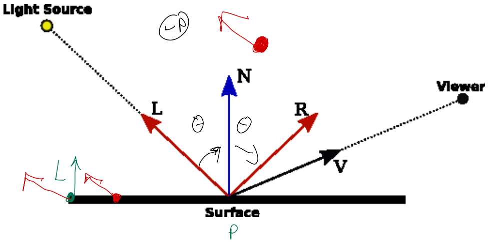


Figure: Components of a directed light.

Notes

- N is the vector normal to the surface.
- R is the direction of the reflected ray.

The Equation Explained Further

Assume the light I has colour components:

- ambient $(I_{a,r}, I_{a,g}, I_{a,b})$;
- diffuse $(I_{d,r}, I_{d,g}, I_{d,b})$; and
- specular $(I_{s,r}, I_{s,g}, I_{s,b})$

and a shininess of m_h. The red vertex colour from the light is

$$I_r \quad = \quad I_{a,r} m_{a,r} + f(I_{d,r} m_{d,r}(L \cdot N) + I_{s,r} m_{s,r} \max(0, V \cdot R)^{m_h})$$

where f is 0 if the surface faces away from the light, else it is 1.

The Equation Explained Further

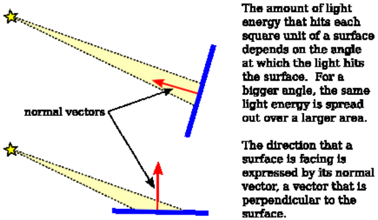


Figure: Energy effect of angle on diffuse light.

Outline

2. Light and Material in OpenGL 1.1
   - Working with Material
   - Defining Normal Vectors
   - Working with Lights
   - Global Lighting Properties

Working with Material

Introduction

Ok, so lets see how all of this light stuff is done in OpenGL.

Turning On The Lights In OpenGL

Definitions (OpenGL functions)

glEnable(GL_LIGHTING) turns on lighting, can be done at anytime except between glBegin and glEnd

glDisable(GL_LIGHTING) turns on lighting, can be done at anytime except between glBegin and glEnd

glEnable(GL_LIGHT0) turns on the mandatory light 0, which shines from the viewer onto the scene. Light is white with no specular component.

Material Attribute Per Vertex

- Material properties are an attribute of a vertex just like colour.
- The current material is stored with each vertex along with its colour and direction.
- Colour, material, directions and light are used to compute the colour at a vertex.
- Since vertexes have a back and front, we need to store two sets per vertex

Material Attributes Per Vertex In OpenGL

Definitions (OpenGL functions)

glMaterialfv(side,property,valueArray) side is the side of the vertex, property is which material property you are setting and valueArray is a list representing (r, g, b, a).

glMaterialf(side,property,value) side is the side of the vertex, property is which material property you are setting and value is a single value property such as shininess.

Definitions (OpenGL constants)

side GL_FRONT_AND_BACK, GL_FRONT, or GL_BACK

property GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR, GL_EMISSION, GL_AMBIENT_AND_DIFFUSE, or GL_SHININESS

Material Attributes Per Vertex In OpenGL

OpenGL C code setting material attributes

```
float bgcolor[4] = { 0.0, 0.7, 0.5, 1.0 };
glMaterialfv( GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE, bgcolor );
```

Note

glEnable(GL_COLOR_MATERIAL) will set the current front and back, ambient and diffuse material properties to the same value as the material colour.

glColorMaterial(side,property) can be used to tweak which side and property.

Defining Normal Vectors

Introduction

The normal vectors for a vertex are essential for lighting calculations. We need to see how exactly this is done in OpenGL.

glNormal3f, glNormal3d, glNormal3fv, and glNormal3dv

OpenGL C code examples of normals

```
glNormal3f( 0, 0, 1 );  // (This is the default value.)
glNormal3d( 0.707, 0.707, 0.0 );

float normalArray[3] = { 0.577, 0.577, 0.577 };
glNormal3fv( normalArray );
```

Notes

- Remember that the normal vector should point out of the front face of the polygon, and that the front face is determined by the order in which the vertices are generated.

Normal For a Cube Face

OpenGL C code example for a cube face

```
glNormal( 0, 1, 0 ); // Points along positive y-axis
glBegin(GL_QUADS);
glVertex3fv(1,1,1);
glVertex3fv(1,1,-1);
glVertex3fv(-1,1,-1);
glVertex3fv(-1,1,1);
glEnd();
```

Notes

- Normals all point in the same direction to give a flat surface.

Normal for a Cylinder

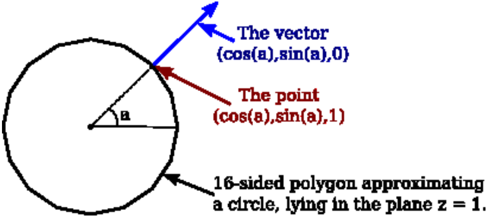


Figure: The normal to the cylinder.

Cylinder Side

OpenGL C code cylinder side

```
glBegin(GL_TRIANGLE_STRIP);
for (i = 0; i <= 16; i++) {
   double angle = 2*3.14159/16 * i;  // i 16-ths of a full circle
   double x = cos(angle);
   double y = sin(angle);
   glNormal3f( x, y, 0 );  // Normal for both vertices at this angle.
   glVertex3f( x, y, 1 );  // Vertex on the top edge.
   glVertex3f( x, y, -1 ); // Vertex on the bottom edge.
}
glEnd();
```

Cylinder Top And Bottom

OpenGL C code cylinder top and bottom

```
glNormal3f( 0, 0, 1);
glBegin(GL_TRIANGLE_FAN);  // Draw the top, in the plane z = 1.
for (i = 0; i <= 16; i++) {
    double angle = 2*3.14159/16 * i;
    double x = cos(angle);
    double y = sin(angle);
    glVertex3f( x, y, 1 );
}
glEnd();

glNormal3f( 0, 0, 1 );
glBegin(GL_TRIANGLE_FAN);  // Draw the bottom, in the plane z = -1
for (i = 16; i >= 0; i--) {
    double angle = 2*3.14159/16 * i;
    double x = cos(angle);
    double y = sin(angle);
    glVertex3f( x, y, -1 );
}
glEnd();
```

glDrawArray And glDrawElements

Definitions (OpenGL functions)

glEnableClientState(GL_NORMAL_ARRAY) enables a normal per a vertex when using glDrawArray and glDrawElements

glNormalPointer(type,stride,data) sets location of the normal data

GL_NORMALIZE

Definitions (OpenGL functions)

glEnable(GL_NORMALIZE) ensures that all normal vectors are unit normals

Warning

Always use GL_NORMALIZE as transformations can affect normals

Working with Lights

Introduction

OpenGL 1.1 supports at least eight light sources. An OpenGL implementation might allow additional lights. Each light source can be configured to be either a directional light or a point light, and each light can have its own diffuse, specular, and ambient intensities.

Eight Lights

- OpenGL 1.1 supports at least eight light sources named
  - GL_LIGHT0, GL_LIGHT1, ...

Definitions (OpenGL functions)

glEnable(light) to enable a light

glLightfv(light,property,valueArray) to set the property values for a light

Eight Lights Cont.

Definitions (OpenGL constants)

- **light** GL_LIGHT0, GL_LIGHT1, ..., GL_LIGHT7
- **property** GL_DIFFUSE, GL_SPECULAR, GL_AMBIENT, or GL_POSITION
- **valueArray** (r, g, b, a) for colour and (x, y, z, w) for lighy

OpenGL C code for a nice blue light

```
float blue1[4] = { 0.4, 0.4, 0.6, 1 };
float blue2[4] = { 0.0, 0, 0.8, 1 };
float blue3[4] = { 0.0, 0, 0.15, 1 };
glLightfv( GL_LIGHT1, GL_DIFFUSE, blue1 );
glLightfv( GL_LIGHT1, GL_SPECULAR, blue2 );
glLightfv( GL_LIGHT1, GL_AMBIENT, blue3 );
```

Directional Versus Point Light

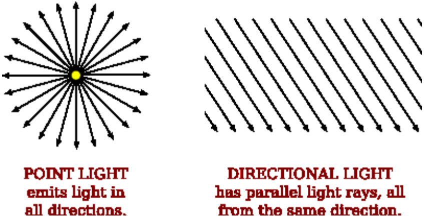


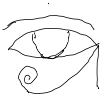
Figure: Point versus directional light.

Note

When $w = 0$ it indicates directional light with direction $(-x, -y, -z)$ and when $w = 1$indicates point light located at (x, y, z). Default position is $(0, 0, 1, 0)$.

Light And Modelview Transform

First, if the position is set before any modelview transformation is applied, then the light is fixed with respect to the viewer. For example, the default light position is effectively set to (0,0,1,0) while the modelview transform is the identity. This means that it shines in the direction of the negative z-axis, in the coordinate system of the viewer, where the negative z-axis points into the screen. Another way of saying this is that the light always shines from the direction of the viewer into the scene. It's like the light is attached to the viewer. If the viewer moves about in the world, the light moves with the viewer.



Light And Modelview Transform

Second, if the position is set after the viewing transform has been applied and before any modelling transform is applied, then the position of the light is fixed in world coordinates. It will not move with the viewer, and it will not move with objects in the scene. It's like the light is attached to the world.

Light And Modelview Transform

Third, if the position is set after a modelling transform has been applied, then the light is subject to that modelling transformation. This can be used to make a light that moves around in the scene as the modelling transformation changes. If the light is subject to the same modelling transformation as an object, then the light will move around with that object, as if it is attached to the object.

Lets See It In Action

Demo

Four Lights Demo

Global Lighting Properties

Introduction

In addition to the properties of individual light sources, the OpenGL lighting system uses several global properties. There are only three such properties in OpenGL 1.1. One of them is the global ambient light.

glLightModel*

Definitions (OpenGL functions)

glLightModelfv(property,value) property must be GL_LIGHT_MODEL_AMBIENT and value is the (r, g, b, a)

glLightModeli(property,value) sets one of the properties below to a value of GL_FALSE and GL_TRUE

Definitions (OpenGL constants)

property GL_LIGHT_MODEL_TWO_SIDE and GL_LIGHT_MODEL_LOCAL_VIEWER

An Example Of Ambient Light

OpenGL C code for a global light

```
glLightModeli( GL_LIGHT_MODEL_TWO_SIDE, 1 ); // Turn on two-sided lighting.

float purple[] = { 0.6, 0, 0.6, 1 };
float yellow[] = { 0.6, 0.6, 0, 1 };
float white[] = { 0.4, 0.4, 0.4, 1 }; // For specular highlights.
float black[] = { 0, 0, 0, 1 };


glMaterialfv( GL_FRONT, GL_AMBIENT_AND_DIFFUSE, purple );
// front material
glMaterialfv( GL_FRONT, GL_SPECULAR, white );
glMaterialf( GL_FRONT, GL_SHININESS, 64 );

glMaterialfv( GL_BACK, GL_AMBIENT_AND_DIFFUSE, yellow );
// back material
glMaterialfv( GL_BACK, GL_SPECULAR, black );  // no specular highlights
```

Lets See That In Action

Demo

Two-sided Lighting

Outline

3. Image Textures
   - Texture Coordinates
   - MipMaps and Filtering
   - Texture Target and Texture Parameters
   - Texture Transformation
   - Loading a Texture from Memory
   - Texture from Colour Buffer
   - Texture Objects
   - Loading Textures in C

Texture Coordinates

Introduction

...

Textures

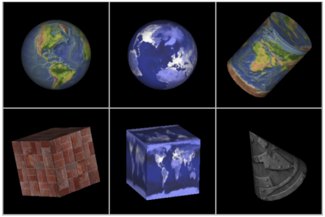


Figure: Some Textures.

Terminology

Definitions

Texture Variation in some property from point-to-point on an object. The most common type is image texture. When an image texture is applied to a surface, the surface color varies from point to point.

Image Texture An image that is applied to a surface as a texture, so that it looks at if the image is "painted" onto the surface.

Terminology Cont.

Definition

Texture coordinates Refers to the coordinate system on a texture image. Texture coordinates typically range from 0 to 1 both vertically and horizontally, with (0,0) at the lower left corner of the image. The term also refers to coordinates that are given for a surface and that are used to specify how a texture image should be mapped to the surface.

Warning

A texture image width and height must be powers of two.

Texture Coordinates

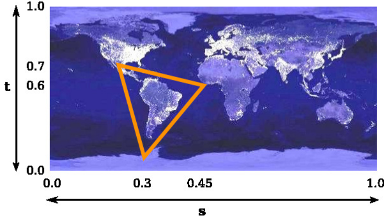


Figure: Texture coordinates t and s.

Using Textures

OpenGL C code using textures on a triangle

```
glNormal3d(0,0,1);        // This normal works for all three vertices.
glBegin(GL_TRIANGLES);
glTexCoord2d(0.3,0.1);    // Texture coords for vertex (0,0)
glVertex2d(0,0);
glTexCoord2d(0.45,0.6);   // Texture coords for vertex (0,1)
glVertex2d(0,1);
glTexCoord2d(0.25,0.7);   // Texture coords for vertex (1,0)
glVertex2d(1,0);
glEnd();
```

Using Textures

OpenGL C code using textures on a rectangle

```
glBegin(GL_TRIANGLE_FAN);
glNormal3f(0,0,1);
glTexCoord2d(0,0);     // Texture coords for lower left corner
glVertex2d(-0.5,-0.5);
glTexCoord2d(1,0);     // Texture coords for lower right corner
glVertex2d(0.5,-0.5);
glTexCoord2d(1,1);     // Texture coords for upper right corner
glVertex2d(0.5,0.5);
glTexCoord2d(0,1);     // Texture coords for upper left corner
glVertex2d(-0.5,0.5);
glEnd();
```

Textures For glDrawArray and glDrawElements

- Follows the standard pattern
- Use glEnableClientState(GL_TEXTURE_COORD_ARRAY) to enable and
- glTexCoordPointer(size,dataType,stride,array) to set the values

MipMaps and Filtering

Introduction

When a texture is applied to a surface, the pixels in the texture do not usually match up one-to-one with pixels on the surface, and in general, the texture must be stretched or shrunk as it is being mapped onto the surface.

Terminology

Definitions

Minification filter An operation that is used when a texture needs to be shrunk when applying it to an object. A minification filter is applied to compute the colour of a pixel when that pixel covers several pixels in the image.

Magnification filter An operation that is used when a texture needs to be stretched when applying it to an object. A magnification filter is applied to compute the colour of a pixel when that pixel covers just a fraction of a pixel in the image.

Terminology

Definitions

Texel A pixel in a texture image.

Mipmap One of a series of reduced-size copies of a texture image, of decreasing width and height. Starting from the original image, each mipmap is obtained by dividing the width and height of the previous image by two (unless it is already 1). The final mimpap is a single pixel. Mipmaps are used for more efficient mapping of the texture image to a surface, when the image has to be shrunk to fit the surface.

Mipmap Example

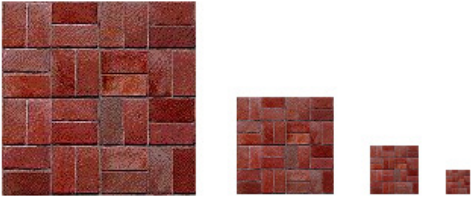


Figure: Example of mipmaps.

Notes

Mipmaps should eventually shrink to one pixel

Texture Target and Texture Parameters

Introduction

OpenGL can actually use 1-D and 3-D textures, as well as 2-D. Because of this, many OpenGL functions dealing with textures take a texture target as a parameter, to tell whether the function should be applied to one, two, or three dimensional textures. For us, the only texture target will be GL_TEXTURE_2D.

glTexParameteri

Definitions (OpenGL functions)

glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR)

glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,minFilter)

default minFilter is GL_NEAREST_MIPMAP_LINEAR

glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_S,GL_CLAMP)

glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_T,GL_CLAMP)

glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_T,GL_REPEAT)

glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_T,GL_REPEAT)

minFilter

Definitions (OpenGL constants)

minFilter GL_NEAREST, GL_LINEAR, GL_NEAREST_MIPMAP_LINEAR, GL_LINEAR_MIPMAP_LINEAR, GL_NEAREST_MIPMAP_NEAREST and GL_LINEAR_MIPMAP_NEAREST

Warning

If you are not using mipmaps for a texture, it is imperative that you change the minification filter for that texture to GL_LINEAR. The default MIN filter requires mipmaps, and if mipmaps are not available, then the texture is considered to be improperly formed, and OpenGL ignores it! Remember that if you don't create mipmaps and if you don't change the minification filter, then your texture will simply be ignored by OpenGL.

Example

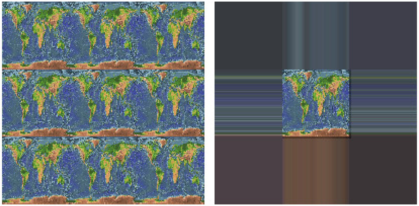


Figure: Repeat versus clamp of the wrap parameter.

Texture Transformation

Introduction

When a texture is applied to a primitive, the texture coordinates for a vertex determine which point in the texture is mapped to that vertex. The texture coordinates are also subject to transformations.

Terminology

Definitions

Texture transformation A transformation that is applied to texture coordinates before they are used to sample data from a texture. The effect is to translate, rotate, or scale the texture on the surface to which it is applied.

OpenGL C code

```
glMatrixMode(GL_TEXTURE);
glLoadIdentity(); // Make sure we are starting from the identity matrix.
glScalef(2,2,1);
glMatrixMode(GL_MODELVIEW); // Leave matrix mode set to GL_MODELVIEW.
```

Lets See That In Action

Demo

Textures and Texture Transforms

Loading a Texture from Memory

Introduction

We been discussing textures. But how exactly do we get a texture into OpenGL.

glTexImage2D

Definitions (OpenGL functions)

glEnable(GL_TEXTURE_2D)
: enables the use of textures, can be disabled

glTexImage2D(GL_TEXTURE_2D,mipmap,GL_RGBA,w,h,b,format,type,pixels)
: **format** tells how the original image data is represented and **pixels** points to the start of the data in memory for **mimmap** level.

Definitions (OpenGL constants)

format
: GL_RGB or GL_RGBA

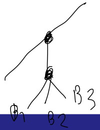
dataType
: GL_UNSIGNED_BYTE

Texture from Colour Buffer

Introduction

Sometimes, instead of loading an image file, it's convenient to have OpenGL create the image internally, by rendering it.

glCopyTexImage2D



Definitions (OpenGL functions)

glCopyTexImage2D(GL_TEXTURE_2D,mipmap,GL_RGBA,x,y,w,h,border)
: specified rectangle from the colour buffer will be copied to texture memory for **mimmap** level.

Lets See It In Action

Demo

Drawing a Texture

Texture Objects

Introduction

Texture objects offer the possibility of storing texture data for multiple textures on the graphics card. With texture objects, you can switch from one texture object to another with a single, fast OpenGL command.

Using Texture Objects

OpenGL C code for working with texture objects

```
int idList[n];
glGenTextures( n, idList ); //create texture id list
for (i = 0; i < n; i++) {
    glBindTexture( iDList[i] );
      .
      .  // Load texture image number i
      .  // Configure texture image number i
      .
}
          .
      .  // Do Some Stuff
      .
glBindTexture( idList[0] ); //Get that texture back
```

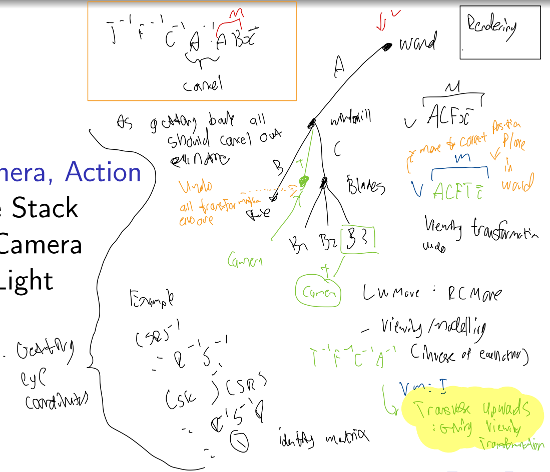
Loading Textures in C

Introduction

See the textbook chapter 4.3.8 for an example of how to use FreeImage to load textures into memory before using glTexImage2D.

Outline

4 Lights, Camera, Action
- Attribute Stack
- Moving Camera
- Moving Light



Attribute Stack

Introduction

glPushMatrix and glPopMatrix are used to manipulate the transform stack. In addition OpenGL 1.1 maintains an attribute stack, which is manipulated using the functions glPushAttrib and glPopAttrib.

glPushAttrib

Definitions (OpenGL functions)

glPushAttrib(GL_ENABLED_BIT) save a copy of each of the OpenGL state variables that can be enabled or disabled

glPushAttrib(GL_CURRENT_BIT) saves a copy of the current colour, normal vector, and texture coordinates

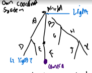
glPushAttrib(GL_LIGHTING_BIT) saves attributes relevant to lighting such as the values of material properties and light properties

glPopAttrib() restores last push

glIsEnabled(name) can be used to test if a state is enabled

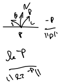
Moving Camera

- Own coordinate system



$F^{-1} B^{-1} X \bar{l}$ $\quad F^{-1} B^{-1} A D \bar{l}_2$ eye coordinate

- Lighting equation in eye coordinates



- Dont transverse light as we find it
- every light in scene transverse

Depth first search

$L2: F^{-1} B^{-1} A B$
$L1: F^{-1} B^{-1} X \bar{l}$

After this transverse and render
(Lighting
↳ Transverse x2

$Lights = [L1, L2]$

Very Important!!!

How Rendering Works

Introduction

...

Moving Light

Introduction

...

David J. Eck; Introduction to Computer Graphics; 2016; http://math.hws.edu/graphicsbook/