Course:
Computer Graphics

Illumination

Book: The read book, OpenGL Superbible

Veronica Orvalho

veronica.orvalho@dcc.fc.up.pt

www.dcc.fc.up.pt/~veronica.orvalho

agenda

1. demos
2. illumination step by step
3. OpenGL example
4. lab 3

videos

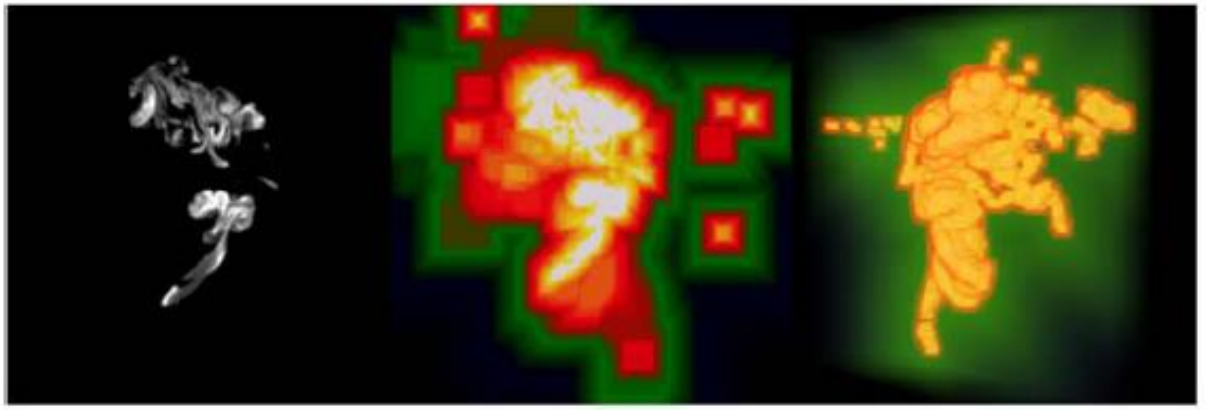
Natural Light Render demonstration in the style of Paul Debevec
http://www.youtube.com/watch?v=sDAYBG6L8HY

Videoman using OpenCV
http://www.youtube.com/watch?v=huFpdL8us0w

Post-production Facial Performance Relighting using Reflecta
http://www.youtube.com/watch?v=FlvXzHLNUS0

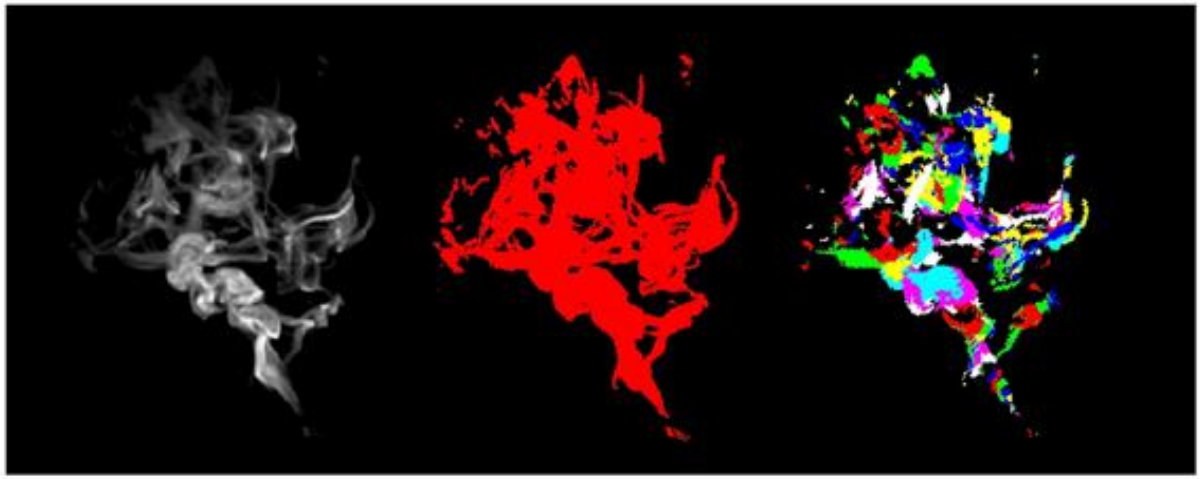
Rapid Acquisition of Specular and Diffuse Normal Maps
http://www.youtube.com/watch?v=Ij5naq3mny0

Paul Devebec presentations (50 minutes)
http://www.youtube.com/watch?v=556FvXHLtAo



Distance map optimization:

Left*:** One of the 256 slices of smoke from a volume data set. ***Middle: Distance map corresponding to this slice. Brighter regions correspond to lower distances. ***Right***: Volumetric representation of the whole distance map



Values generated by pixel validity mask pass. From ***left* to *right***: Smoke volume being rendered, pixel validity mask showing *red* pixels for those fragments containing valid information and color representation of index of the first slice containing non-transparent data (*iFirst*)

Composition

***Left to right*:** Simple back-to-front alpha compositing, primary rays, primary rays and shadow rays

"Interactive HDR lighting of dynamic participating media" by F. Navarro, D. Guitierrez, F. Serón, Vis Comput (2009)

***left-right* and *top-bottom* order.**

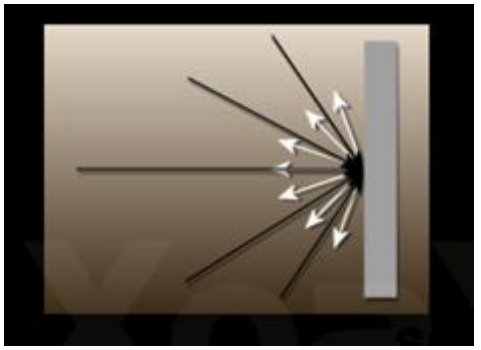
Smoke rendered using eight different light probes

“Interactive HDR lighting of dynamic participating media” by F. Navarro, D. Guitierrez, F. Serón, Vis Comput (2009)

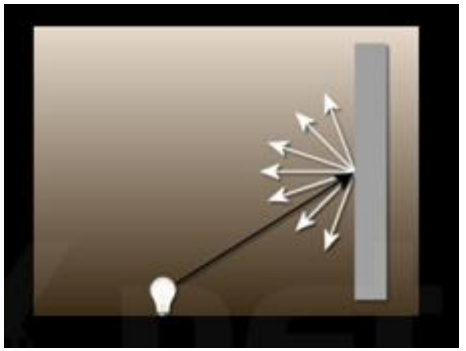
some basics you MUST know

Types of Lights

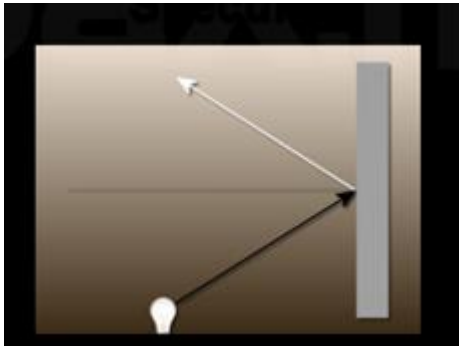
Ambient:
No source point; affects all polys independent of position, orientation and viewing angle; used as a ‘fudge’ to approximate 2nd order and higher reflections



Diffuse:
Light scattered in all directions after it hits a poly; dependant upon incident angle



Specular:
‘Shininess’ ; dependant upon incident and viewing angles



Images: http://xoax.net/comp/sci/graphics3D/BasLocIll.php

some basics you MUST know

Types of Lights

1. **Ambient**
2. **Diffuse**
3. **Specular**

4. Emissive: color of a surface adds intensity to the object, but is unaffected by any light sources. Does not introduce any additional light into the overall scene.

Images: http://xoax.net/comp/sci/graphics3D/BasLocIll.php

some basics you MUST know

Implementation Specifications

1. Two kinds of parameters: lighting and material

2. Material properties: state variables, can be changed as you draw different polys in a scene

3. Light properties: parameters indexed to light numbers;
OpenGL can use up to **8 lights**;
light positions are affected by the modelview matrix stack

some basics you MUST know

Shading Model

defines how the lighting equations are applied to a rasterized poly.

GL_FLAT: Lighting is evaluated once per poly, and the resulting color value is used for the whole object.

GL_SMOOTH: Lighting is evaluated at each vertex, and pixel colors are linearly interpolated across polys. This is more expensive, but it looks much better.

OpenGL uses the **Phong** lighting model at vertices, but has no built-in support for Phong shading.

some basics you MUST know

Normals

1. The lighting equations depend upon normals. We need to provide them.

2. The current normal can be specified **glNormal*** function, and will be applied to every subsequent vertex.
(or you can load them from a file)

3. Normals should be of unit length, or the lighting equations will not work correctly. This can be a problem, because normals are affected by any scaling done in the matrix stack. You must either re-normalize the normals as a pre-processing step, or enable **GL_NORMALIZE** (which is computationally expensive).

example illumination in OpenGL

How can I make my light position stay fixed relative to my eye position?

Specify your light in eye coordinate space:

- set the ModelView matrix to the identity
- then specify your light position.

How do I make a headlight?

a light that appears to be positioned at or near the eye and shining along the line of sight:

- set the ModelView to the identity,
- set the light position at (or near) the origin,
- and set the direction to the negative Z axis.

OpenGL FAQ

opengl.org

example illumination in OpenGL

How can I make my light stay fixed relative to my scene?

As your view changes, your ModelView matrix also changes. This means you'll need to respecify the light position, usually at the start of every frame. A typical application will display a frame with the following pseudo-code:

1. Set the view transform.
2. Set the light position
3. Send down the scene or model geometry.
5. Swap buffers.

OpenGL FAQ
opengl.org

example illumination in OpenGL

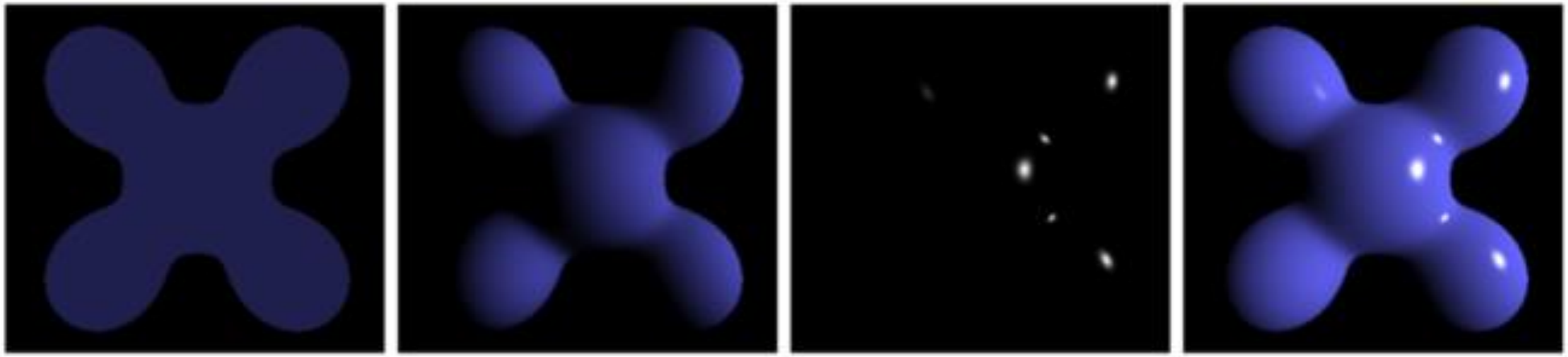
How can I make a light that moves around in a scene?
you'll need to respecify this light position every time the view changes. Additionally, this light has a dynamic modeling transform that also needs to be in the ModelView matrix before you specify the light position. In pseudo-code, you need to do something like:

1. Set the view transform
2. Push the matrix stack
3. Set the model transform to update the light's position
4. Set the light position
5. Pop the matrix stack
6. Send down the scene or model geometry
7. Swap buffers

OpenGL FAQ
opengl.org

some basics you MUST know

Types of Lights

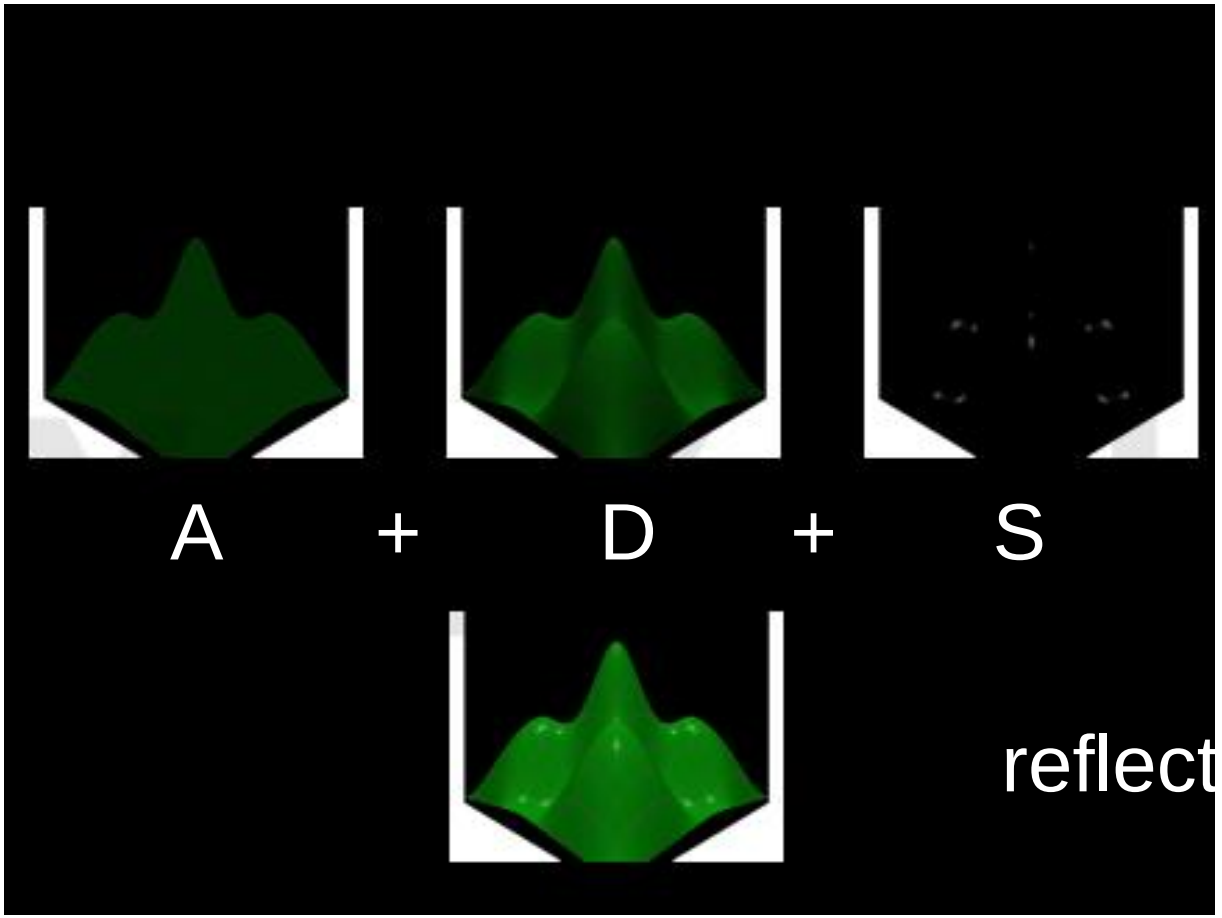


ambient + diffuse + specular = phong

reflection

some basics you MUST know

Types of Lights



Tutorial (video):
http://xoax.net/comp/sci/graphics3D/BasLocIll.php

example illumination in OpenGL

OpenGL init code

```
//set the global lighting / shading params
glShadeModel(GL_SMOOTH); // or GL_FLAT
glEnable(GL_NORMALIZE); //or not
glEnable(GL_LIGHTING);

//set the global ambient light
GLfloat ambient = {.2,.2,.2,1};
glLightModelfv(GL_LIGHT_MODEL_AMBIENT, globalAmb);

//This code sets up a light and enables it
GLfloat diffuse[] = {1,0,0,1};
GLfloat ambient[] = {.5,0,0,1};
GLfloat specular[] = {1,1,1,1};

glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuse);
glLightfv(GL_LIGHT0, GL_AMBIENT, ambient);
glLightfv(GL_LIGHT0, GL_SPECULAR, specular);
glEnable(GL_LIGHT0); //enable the light
```

example illumination in OpenGL

OpenGL init code

```
//set the global lighting / shading params
glShadeModel(GL_SMOOTH); // or GL_FLAT
glEnable(GL_NORMALIZE); //or not
glEnable(GL_LIGHTING);

//set the global ambient light
GLfloat ambient = {.2,.2,.2,1};
glLightModelfv(GL_LIGHT_MODEL_AMBIENT, globalAmb);

//This code sets up a light and enables it
GLfloat diffuse[] = {1,0,0,1};
GLfloat ambient[] = {.5,0,0,1};
GLfloat specular[] = {1,1,1,1};

glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuse);
glLightfv(GL_LIGHT0, GL_AMBIENT, ambient);
glLightfv(GL_LIGHT0, GL_SPECULAR, specular);
glEnable(GL_LIGHT0); //enable the light
```

example illumination in OpenGL

optimize code (init)

```
//we need to store our lights as we might have more than 1
typedef struct
{
  float pos[4];
  float diffuse[4];
  float specular[4];
  float ambient[4];
} light_t;

//instance of a light
light_t light={
     {6,10,15,1},  //position (the final 1 means the light is positional)
     {1,1,1,1},    //diffuse
     {0,0,0,1},    //specular
     {0,0,0,1}     //ambient
   };
```

example illumination in OpenGL

optimize code (Render)

```
//redraw function
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

glLightfv(GL_LIGHT0,GL_POSITION,light.pos);       //updates the light's position
glLightfv(GL_LIGHT0,GL_DIFFUSE,light.diffuse);    //updates the light's diffuse color
glLightfv(GL_LIGHT0,GL_SPECULAR,light.specular);  //updates the light's specular color
glLightfv(GL_LIGHT0,GL_AMBIENT,light.ambient);    //updates the light's ambient color

drawObject(...);
```

example: ambient light

```
// . A white bright light is used
// . All objects are illuminated evenly from all sides
#define glRGB(x, y, z) glColor3ub((GLubyte)x, (GLubyte)y, (GLubyte)z)

void setup()
{
// Light values
// Bright white light
GLfloat ambientLight[] = { 1.0f, 1.0f, 1.0f, 1.0f };

glEnable(GL_DEPTH_TEST);      // Hidden surface removal
glEnable(GL_CULL_FACE);       // Do not calculate inside of jet
glFrontFace(GL_CCW);          // Counter clock-wise polygons face out

// Lighting stuff
glEnable(GL_LIGHTING);                  // Enable lighting

// Set light model to use ambient light specified by ambientLight
// all objects are illuminated evenly from all sides
glLightModelfv(GL_LIGHT_MODEL_AMBIENT,ambientLight);


glEnable(GL_COLOR_MATERIAL);      // Enable Material color tracking

// Front material ambient and diffuse colors track glColor
// all primitives after this call are affected by the last value set,
// until another call to glMaterial is made
glColorMaterial(GL_FRONT,GL_AMBIENT_AND_DIFFUSE);

// Light blue background
glClearColor(0.0f, 0.0f, 05.f,1.0f);
}
```

ambient+diffuse+light position

```
void setup()
{
// Light values and coordinates
GLfloat  ambientLight[] = { 0.3f, 0.3f, 0.3f, 1.0f };
GLfloat  diffuseLight[] = { 0.7f, 0.7f, 0.7f, 1.0f };
GLfloat       lightPos[] = { -50.f, 50.0f, 100.0f, 1.0f };

glEnable(GL_DEPTH_TEST);       // Hidden surface removal
glFrontFace(GL_CCW);           // Counter clock-wise polygons face out
glEnable(GL_CULL_FACE);        // Do not calculate inside of jet

// Enable lighting
glEnable(GL_LIGHTING);

// Setup and enable light 0
glLightfv(GL_LIGHT0,GL_AMBIENT,ambientLight);
glLightfv(GL_LIGHT0,GL_DIFFUSE,diffuseLight);
glLightfv(GL_LIGHT0,GL_POSITION,lightPos);
glEnable(GL_LIGHT0);

// Enable color tracking
glEnable(GL_COLOR_MATERIAL);

// Set Material properties to follow glColor values
glColorMaterial(GL_FRONT, GL_AMBIENT_AND_DIFFUSE);

// Light blue background
glClearColor(0.0f, 0.0f, 1.0f, 1.0f );
}
```

modify this code to add **SPECULAR**

```
void setup()
{
// Light values and coordinates
GLfloat  ambientLight[] = { 0.3f, 0.3f, 0.3f, 1.0f };
GLfloat  diffuseLight[] = { 0.7f, 0.7f, 0.7f, 1.0f };
GLfloat       lightPos[] = { -50.f, 50.0f, 100.0f, 1.0f };

glEnable(GL_DEPTH_TEST);       // Hidden surface removal
glFrontFace(GL_CCW);           // Counter clock-wise polygons face out
glEnable(GL_CULL_FACE);        // Do not calculate inside of jet

// Enable lighting
glEnable(GL_LIGHTING);

// Setup and enable light 0
glLightfv(GL_LIGHT0,GL_AMBIENT,ambientLight);
glLightfv(GL_LIGHT0,GL_DIFFUSE,diffuseLight);
glLightfv(GL_LIGHT0,GL_POSITION,lightPos);
glEnable(GL_LIGHT0);

// Enable color tracking
glEnable(GL_COLOR_MATERIAL);

// Set Material properties to follow glColor values
glColorMaterial(GL_FRONT, GL_AMBIENT_AND_DIFFUSE);

// Light blue background
glClearColor(0.0f, 0.0f, 1.0f, 1.0f );
}
```

specular

```
void setup()
{
// Light values and coordinates
GLfloat  ambientLight[] = { 0.3f, 0.3f, 0.3f, 1.0f };
GLfloat  diffuseLight[] = { 0.7f, 0.7f, 0.7f, 1.0f };
GLfloat  specular[] = { 1.0f, 1.0f, 1.0f, 1.0f };
GLfloat  lightPos[] = { -50.f, 50.0f, 100.0f, 1.0f };

glEnable(GL_DEPTH_TEST);      // Hidden surface removal
glFrontFace(GL_CCW);          // Counter clock-wise polygons face out
glEnable(GL_CULL_FACE);       // Do not calculate inside of jet

// Enable lighting
glEnable(GL_LIGHTING);

// Setup and enable light 0
glLightfv(GL_LIGHT0,GL_AMBIENT,ambientLight);
glLightfv(GL_LIGHT0,GL_DIFFUSE,diffuseLight);
glLightfv(GL_LIGHT0,GL_DIFFUSE,specular);
glLightfv(GL_LIGHT0,GL_POSITION,lightPos);
glEnable(GL_LIGHT0);

// Enable color tracking
glEnable(GL_COLOR_MATERIAL);

// Set Material properties to follow glColor values
glColorMaterial(GL_FRONT, GL_AMBIENT_AND_DIFFUSE);

// Light blue background
glClearColor(0.0f, 0.0f, 1.0f, 1.0f );
}
```

render example

```
void RenderScene(void)
{
// Set material color
glRGB(0, 255, 0);
glBegin(GL_TRIANGLES);
  glNormal3f(0.0f, -1.0f, 0.0f);
  glVertex3f(0.0f, 0.0f, 60.0f);
  glVertex3f(-15.0f, 0.0f, 30.0f);
  glVertex3f(15.0f,0.0f,30.0f);
glEnd();


// Verticies for this panel
float v[3][3] =      {{ 15.0f, 0.0f, 30.0f},
{ 0.0f, 15.0f, 30.0f},
{ 0.0f, 0.0f, 60.0f}};

// Calculate the normal for the plane
calcNormal(v,normal);

// Draw the triangle using the plane normal
// for all the vertices
glBegin(GL_TRIANGLES);
  glNormal3fv(normal);
  glVertex3fv(v[0]);
  glVertex3fv(v[1]);
  glVertex3fv(v[2]);
glEnd();

}
```

compute normal

```
// Points p1, p2, & p3 specified in counter clock-wise order
void calcNormal(float v[3][3], float out[3])
{
float v1[3],v2[3];
static const int x = 0;
static const int y = 1;
static const int z = 2;
// Calculate two vectors from the three points
v1[x] = v[0][x] - v[1][x];
v1[y] = v[0][y] - v[1][y];
v1[z] = v[0][z] - v[1][z];

v2[x] = v[1][x] - v[2][x];
v2[y] = v[1][y] - v[2][y];
v2[z] = v[1][z] - v[2][z];
// Take the cross product of the two vectors to get
// the normal vector which will be stored in out
out[x] = v1[y]*v2[z] - v1[z]*v2[y];
out[y] = v1[z]*v2[x] - v1[x]*v2[z];
out[z] = v1[x]*v2[y] - v1[y]*v2[x];

// Normalize the vector (shorten length to one)
ReduceToUnit(out);
}
```

compute unit vector

```
// Reduces a normal vector specified as a set of three coordinates,
// to a unit normal vector of length one.
void ReduceToUnit(float vector[3])
{
float length;

// Calculate the length of the vector
length = (float)sqrt((vector[0]*vector[0]) +
(vector[1]*vector[1]) +
(vector[2]*vector[2]));

// Keep the program from blowing up by providing an exceptable
// value for vectors that may calculated too close to zero.
if(length == 0.0f)
length = 1.0f;

// Dividing each element by the length will result in a
// unit normal vector.
vector[0] /= length;
vector[1] /= length;
vector[2] /= length;
}
```

specular reflectance

```
void setup()
{
// Light values and coordinates
GLfloat  ambientLight[] = { 0.3f, 0.3f, 0.3f, 1.0f };
GLfloat  diffuseLight[] = { 0.7f, 0.7f, 0.7f, 1.0f };
GLfloat  specular[] = { 1.0f, 1.0f, 1.0f, 1.0f };
GLfloat  lightPos[] = { -50.f, 50.0f, 100.0f, 1.0f };
GLfloat  specReflectance[] = { 1.0f, 1.0f, 1.0f, 1.0f };

...

// Enable lighting
glEnable(GL_LIGHTING);

// Setup and enable light 0
...

// Enable color tracking
glEnable(GL_COLOR_MATERIAL);

// Set Material properties to follow glColor values
glColorMaterial(GL_FRONT, GL_AMBIENT_AND_DIFFUSE);

// All materials hereafter have full specular reflectivity with high shine
glMaterialfv(GL_FRONT, GL_SPECULAR, specReflectance);
glMateriali(GL_FRONT, GL_SHININESS, 128);


// Light blue background
glClearColor(0.0f, 0.0f, 1.0f, 1.0f );
}
```

spotlight

```
void setup()
{
// Light values and coordinates
GLfloat  ambientLight[] = { 0.3f, 0.3f, 0.3f, 1.0f };
GLfloat  diffuseLight[] = { 0.7f, 0.7f, 0.7f, 1.0f };
GLfloat  specular[] = { 1.0f, 1.0f, 1.0f, 1.0f };
GLfloat  lightPos[] = { -50.f, 50.0f, 100.0f, 1.0f };
GLfloat  specReflectance[] = { 1.0f, 1.0f, 1.0f, 1.0f };
...
// Enable lighting
glEnable(GL_LIGHTING);
// Setup and enable light 0

// Setup and enable light 0
glLightfv(GL_LIGHT0,GL_AMBIENT,ambientLight);
glLightfv(GL_LIGHT0,GL_DIFFUSE,diffuseLight);
// Specific spot effect, cut off angle is 60 degree
glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, 60.0f);
// Shiny spot
glLightf(GL_LIGHT0, GL_SPOT_EXPONENT, 100.0f);
...
// Set Material properties to follow glColor values
glColorMaterial(GL_FRONT, GL_AMBIENT_AND_DIFFUSE);

// All materials hereafter have full specular reflectivity with high shine
glMaterialfv(GL_FRONT, GL_SPECULAR, specReflectance);
glMateriali(GL_FRONT, GL_SHININESS, 128);

// Light blue background
glClearColor(0.0f, 0.0f, 1.0f, 1.0f );
}
```

shadows

shadows appear when an object **keeps light** from a light source from striking some object or surface behind the object

shadows

shadows appear when an object **keeps light** from a light source from striking some object or surface behind the object

how do you create a shadow?

shadows

1. flatten the Modelview projection matrix, so all objects are on a 2D world. It is squished to the plane where it lies.

shadows

1. flatten the Modelview projection matrix, so all objects are on a 2D world. It is squished to the plane where it lies.

2. calculate the distance and direction of the light source. It determines the shape of and influences the size of the shadow.

shadows

1. flatten the Modelview projection matrix, so all objects are on a 2D world. It is squished to the plane where it lies.

2. calculate the distance and direction of the light source. It determines the shape of and influences the size of the shadow.

You will need to create advance matrix, taking into consideration the shadow plane and light position

Assignment 2: illumination + textures

Due: April 13th at 24h00

1. implement the equations of lambert, phong, gouraud and apply it to an object
(eg. sphere, terrain)

2 support at least 3 light sources (ambient, point, directional)

3 use the .OBJ file from assignment 1 to apply the lambert shading model and load the textures

4 create different lights using OpenGL* (at least 3 different lights)

5 implement planar reflections (you can use OpenGL geometric transformations)

6 implement texture mapping using OpenGL* and test the different filters (at least 2 types of filters)

7 allow manipulating (eg. Keyboard) the different lights: type of light, position, intensities, colors)

* Use OpenGL or the API of your choice, shaders.

references

Tutorial on Lighting in OpenGL:

http://glprogramming.com/red/chapter05.html (the Read Book)
http://www.swiftless.com/tutorials/opengl/lighting.html
http://www.falloutsoftware.com/tutorials/gl/gl8.htm
http://www.gamedev.net/reference/list.asp?categoryid=31
http://nehe.gamedev.net/data/lessons/lesson.asp?lesson=07
http://www.sulaco.co.za/tut3.htm

http://xoax.net/comp/sci/graphics3D/BasLocIll.php (video)