Lab 3
Shaders

CSE 4431/5331.03M

Advanced Topics in 3D Computer Graphics

TA: Margarita Vinnikov

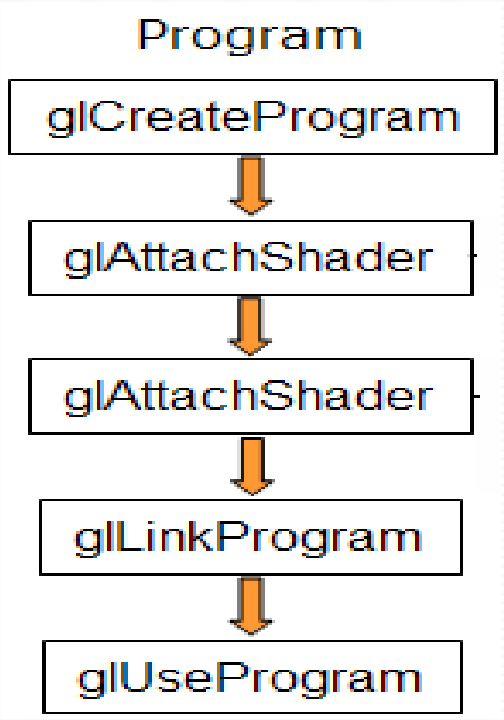
mvinni@cse.yorku.ca

Shaders

What do we need to do to use shaders in our program ?

Shaders – Work flow

- For multiple shaders
  1. Create program
  2. Attach shader object to appropriate program
  3. Link the shader
  4. Verify the shader program
  5. Verify that link was successful
  6. Use the shader for vertex or fragment processing



http://www.lighthouse3d.com/tutorials/glsl-tutorial/creating-a-program/

```
void setShaders() {
    char *vs = NULL,*fs = NULL,*fs2 = NULL;
    //Create program
    v = glCreateShader(GL_VERTEX_SHADER);
    f = glCreateShader(GL_FRAGMENT_SHADER);
    //read shaders' source code
    vs = textFileRead("toon.vert");
    fs = textFileRead("toon.frag");
    const char * ff = fs;
    const char * ff2 = fs2;
    //connect with the source
    glShaderSource(v, 1, &vv,NULL);
    glShaderSource(f, 1, &ff,NULL);
    free(vs);free(fs);
    glCompileShader(v);
    glCompileShader(f);
    p = glCreateProgram();
    glAttachShader(p,f);
    glAttachShader(p,f2);
    glLinkProgram(p);
    glUseProgram(p); /// activate the program when rendering
}
```

```
GLuint v,f,f2,p;
glewInit()
{
    …
    setShaders();
    glutMainLoop();
    …
}
char *textFileRead(char *fn)
{
    FILE *fp;
    char *content = NULL;
    int count=0;
    if (fn != NULL)
    {
      fp = fopen(fn,"rt");
      if (fp != NULL)
      {
        fseek(fp, 0, SEEK_END);
        count = ftell(fp);
        rewind(fp);
        if (count > 0)
        {
          content = (char*)malloc(sizeof
                    (char) * (count+1));
          count = fread(content,sizeof
                  (char),count,fp);
          content[count] = '\0';
        }
        fclose(fp);
    } } return content;}
```

http://www.lighthouse3d.com/tutorials/glsl-tutorial/creating-a-program/

Debugging Shaders

```
void setShaders()
{

  …

  glCompileShader(v);
  glCompileShader(f);

  //verify successful
  compilation
  printShaderInfoLog(vShader);
  printShaderInfoLog(fShader);

  …
}
```

```
void printShaderInfoLog(GLuint obj)
{
 int infologLength = 0;
 int charsWritten  = 0;
 char *infoLog;
 glGetShaderiv(obj,
     GL_INFO_LOG_LENGTH, &infologLength);
 if (infologLength > 0)
 {
   infoLog = (char *)malloc(infologLength);
   glGetShaderInfoLog(obj, infologLength,
              &charsWritten, infoLog);
   printf("%s\n",infoLog);
   free(infoLog);
  }
}
```

Debugging Sahaders

```
void setShaders()
{
 …
 glLinkProgram(p);
 printProgramInfoLog(p);


}
```

```
void printProgramInfoLog(GLuint obj)
{
    int infologLength = 0;
    int charsWritten  = 0;
    char *infoLog;

    glGetProgramiv(obj,
        GL_INFO_LOG_LENGTH,&infologLength);
    if (infologLength > 0)
    {
      infoLog = (char *)malloc
                              (infologLength);
      glGetProgramInfoLog(obj,
                             infologLength,
                    &charsWritten, infoLog);
      printf("%s\n",infoLog);
      free(infoLog);
    }
}
```

Debugging Shaders

```
void setShaders()
{
…
   glLinkProgram(p);
   if (checkLinkedProgram(program)
     glUseProgramObject(program);

}
```

```
GLint checkLinkedProgram(Gluint
   program)
 {
     GLint linked;
     glGetShaderiv(program,
      GL_LINK_STATUS, &linked);
     if (!linked)
    {
       GLint length;
       GLchar * log;
       glGetShaderiv(program,

          GL_INFO_LOG_LENGTH,
                    &length);
       log = (GLchar *)malloc(length);
       glGetShaderInfoLog(program,
                          length,
                          &length,
                          log);
       printf("linked log = '%s \n",
          log);
    }
     return linked;
 }
```

```
void glDeleteShader(GLuint id);

void glDeleteProgram(GLuint id);
```

Shaders II

What do we do with shaders?

Simple Vertex Shader

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

OR

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

OR

```
void main() {
  gl_Position = ftransform();
}
```

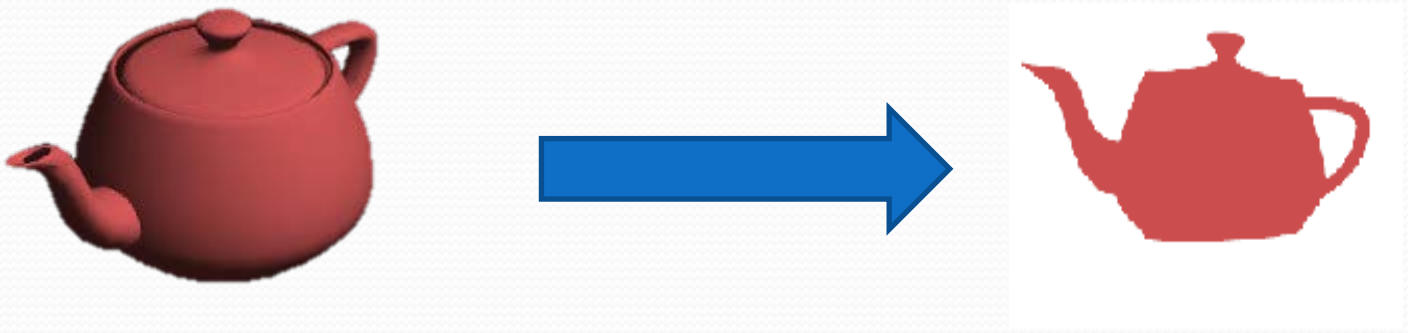
Simple Fragment Shader

```
void main() {
  gl_FragColor = vec4(0.4,0.4,0.8,1.0);
}
OR
void main() {
  gl_FragColor = gl_Color;
}
```

Flatten Vertex Shader

To flatten a 3D model, set z coordinate to zero prior to applying the modelview transformation.

```
void main()
{
    vec4 v = vec4(gl_Vertex);
    v.z = 0.0;
    gl_Position = gl_ModelViewProjectionMatrix * v;
}
```

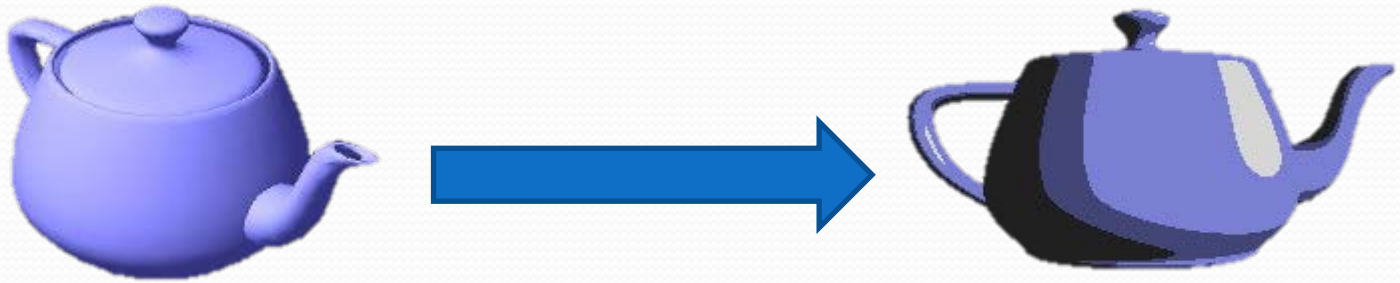


Cartoon Shader

- Large areas of constant color with sharp transitions between them.

Vertex Shader:

```
varying float intensity;
void main() {
  vec3 ld;
    vec3 lightDir = vec3(1,0,0);
    intensity = dot(lightDir,gl_Normal);
    gl_Position = ftransform();
}
```



Cartoon Shader

Fragment Shader:

```
varying float intensity;
void main() {
  vec4 color;
   if (intensity > 0.95)
    color = vec4(1.0,0.5,0.5,1.0);
   else if (intensity > 0.5)
     color = vec4(0.6,0.3,0.3,1.0);
   else if (intensity > 0.25)
     color = vec4(0.4,0.2,0.2,1.0);
   else color = vec4(0.2,0.1,0.1,1.0);
   gl_FragColor = color;
}
```

First task

- Use the code to read and instantiate shaders
- Implement simple shaders
- Implement flatten shader
  - Change the vertices such that z = sin(5.0*x)*0.25
  - Bonus: come up with an interesting perturbation
- Implement cartoon shader
  - Modify light vector and intensity gradation in an interesting way

Vertex animation

- Keep track of time, or a frame counter
  - But a vertex shader can't keep track of values between vertices, let alone between frames.
  - define time variable in the OpenGL application,
  - pass it to the shader as a uniform variable.

In setup:

```
loc = glGetUniformLocation(p,"time");

 void renderScene(void) {
 ...
 glUniform1f(loc, time);
 //draw your model
 glutSolidTeapot(1);
```

Second task

- Modify your code so that you animate either the flatten shader or cartoon shader
- Bonus: do both
- Bonus: make creative animation

Lighting

Check point

Do you have a Virtual Light Source?

- **point source** - shines light equally in all directions.
- It needs:
  - a 3d world position
  - rgb colours
    - L_s, for its specular light colour
    - L_d, for its diffuse light colour
    - L_a, for its ambient light colour
- You can create these 4 variables directly in your shader, but if the light needs to move or change colour then you should use **uniform** variables instead.

Do you have Surface Properties?

- Every object in your scene should have some surface properties.
- These properties define the factor of each lighting approximation that should be reflected by the surface.
- your object can have its own, unique
  - 3d world position
  - rgb colours
    - K_s, for its specular reflectance factor
      - rough surface should have a very low specular reflectance factor
      - A shiny, smooth, surface will have a high specular reflectance factor
    - K_d, for its diffuse reflectance factor
      - The diffuse reflectance factor is usually used to give the object an unique, base colour
    - K_a, for its ambient reflectance factor
    - a surface **normal**; the facing direction of the surface

Check Point

- Did you implement light source in your code?
  - You can rotate the light source
- Do you have surface properties setup for your models?

❑ If not see additional material for the lab. I expect light implemented when you submit the lab.

Quiz

- Ambient

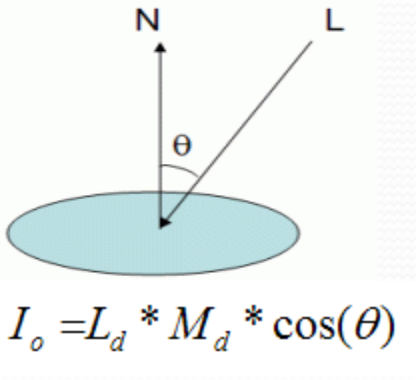
- *Ambient + Diffuse*

- *Specular*

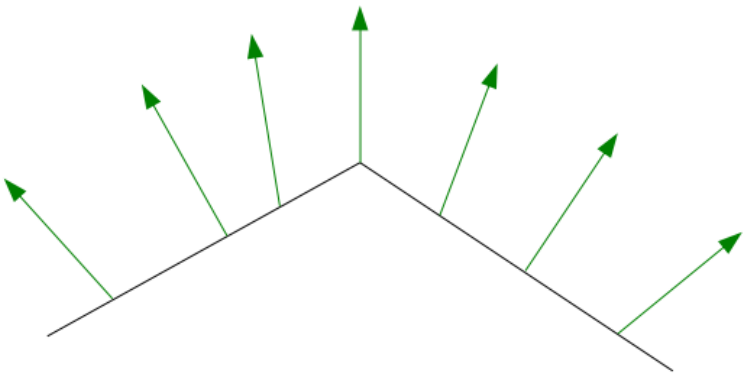
- Point Light

- Spot Light

Diffuse Light



I_o is the reflected intensity,
L_d is the light's diffuse color (*gl_LightSource[o].diffuse*),
M_d is the material's diffuse coefficient (*gl_FrontMaterial.diffuse*).



1. The normal vector and the light direction vector (*gl_LightSource[o].position*) has to be normalized

2. Cos(θ) = lightDir . normal / (|lightDir| * |normal|)
3. Note that we want *gl_Normal* and *lightDir* to be normalized
   | normal | = 1 | lightDir | = 1

   So that
   Cos(θ) = lightDir . normal

OpenGL stores the lights direction in eye space coordinates; hence we need to transform the normal to eye space in order to compute the dot product.

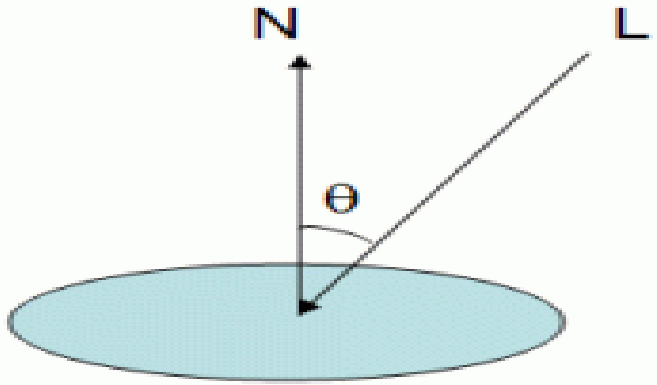
Eye Space

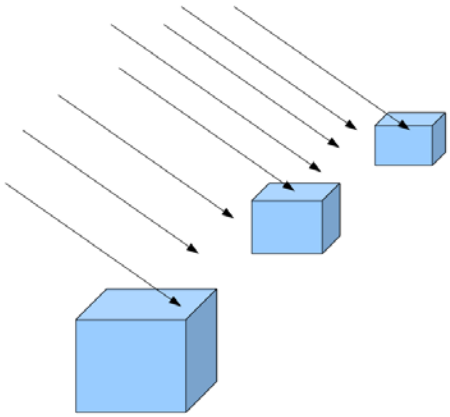
- To transform the normal to eye space use the *gl_NormalMatrix*.
  - This matrix is the transpose of the inverse of the 3x3 upper left sub matrix from the modelview matrix.

```
varying vec3 normal;
void main()
{
  normal = gl_NormalMatrix * gl_Normal;
   …;
}
```

Ambient Light

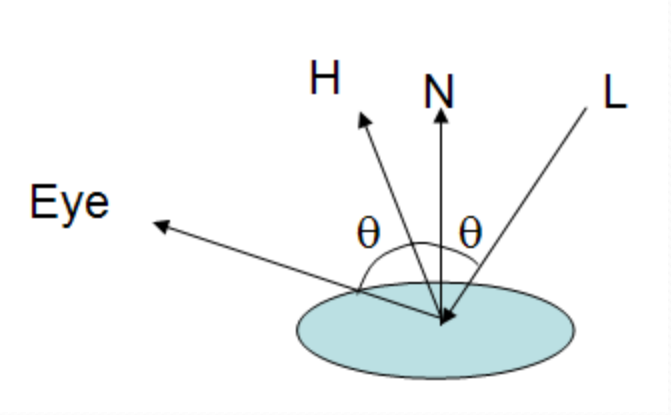
I_a is the ambient intensity,
M_a is the global ambent (gl_LightModel.ambient)
L_a is the light's ambinent color (*gl_LightSource[o].ambient*),
M_a is the material's ambient coefficient (*gl_FrontMaterial.ambient*),





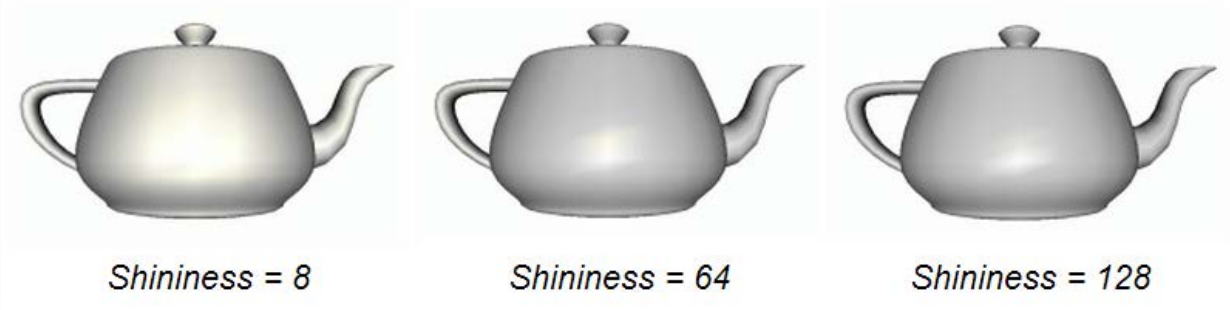
$$I_a = G_a * M_a + L_a * M_a$$

Specular Lights



$$H = Eye - L$$

$$Spec = (N \cdot H)^{s} * L_s * M_s$$



Shininess = 8 Shininess = 64 Shininess = 128

Your Turn

- Follow instructions attached implement light shaders
- Bonus: create an interesting light shader

- Write you name and student number in comments at the top of your code
- Duplicate your .cpp
- Name it lab3_First_Last.cpp
- Submit your code as follows:

    submit 4431 lab1 filename(s)

- http://ogldev.atspace.co.uk/
- http://www.lighthouse3d.com/tutorials/glsl-tutorial/