Grafică pe calculator

Elemente de grafică 3_D

Programare OpenGL 3 (Part c)
- Introduction
- Rendering Primitives
- Rendering Modes
- Lighting
  - Texture Mapping
  - Additional Rendering Attributes
  - Imaging
ANIMATION AND DEPTH BUFFERING

- Discuss double buffering and animation
- Discuss hidden surface removal using the depth buffer
Double Buffering

Front Buffer

Back Buffer

Display
Animation Using Double Buffering

1. Request a double buffered color buffer
   
   ```c
   glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE);
   ```

2. Clear color buffer
   
   ```c
   glutClear(GL_COLOR_BUFFER_BIT);
   ```

3. Render scene

4. Request swap of front and back buffers
   
   ```c
   glutSwapBuffers();
   ```

- Repeat steps 2 - 4 for animation
Depth Buffering and Hidden Surface Removal
Depth Buffering Using OpenGL

1. Request a depth buffer
   
   ```c
   glutInitDisplayMode( GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH );
   ```

2. Enable depth buffering
   
   ```c
   glEnable( GL_DEPTH_TEST );
   ```

3. Clear color and depth buffers
   
   ```c
   glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
   ```

4. Render scene

5. Swap color buffers
void main( int argc, char** argv )
{
    glutInit( &argc, argv );
    glutInitDisplayMode( GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH );
    glutCreateWindow( "Tetrahedron" );
    init();
    glutIdleFunc( idle );
    glutDisplayFunc( display );
    glutMainLoop();
}
void init( void )
{
    glClearColor( 0.0, 0.0, 1.0, 1.0 );
}

void idle( void )
{
    glutPostRedisplay();
}
void drawScene( void )
{
    GLfloat vertices[] = { ... };
    GLfloat colors[] = { ... };
    glClear( GL_COLOR_BUFFER_BIT |
             GL_DEPTH_BUFFER_BIT );
    glBegin( GL_TRIANGLE_STRIP );
    /* calls to glColor*() and glVertex*() */
    glEnd();
    glutSwapBuffers();
}
Lighting simulates how objects reflect light

- material composition of object
- light’s color and position
- global lighting parameters
  - ambient light
  - two sided lighting
- available in both color index and RGBA mode
How OpenGL Simulates Lights

- **Phong lighting model**
  - Computed at vertices

- **Lighting contributors**
  - Surface material properties
  - Light properties
  - Lighting model properties
Surface Normals

- Normals define how a surface reflects light

```cpp
glNormal3f( x, y, z )
```

- Current normal is used to compute vertex's color
- Use *unit* normals for proper lighting
- Scaling affects a normal's length

```cpp
glEnable( GL_NORMALIZE )
```

Or

```cpp
glEnable( GL_RESCALE_NORMAL )
```
Define the surface properties of a primitive

```
glMaterialfv( face, property, value );
```

- GL_DIFFUSE: Base color
- GL_SPECULAR: Highlight Color
- GL_AMBIENT: Low-light Color
- GL_EMISSION: Glow Color
- GL_SHININESS: Surface Smoothness

- separate materials for front and back
glLightfv( light, property, value );

- **light** specifies which light
  - multiple lights, starting with GL_LIGHT0
    
    ```c
    glGetIntegerv( GL_MAX_LIGHTS, &n );
    ```

- **properties**
  - colors
  - position and type
  - attenuation
Light Sources (cont.)

- **Light color properties**
  - `GL_AMBIENT`
  - `GL_DIFFUSE`
  - `GL_SPECULAR`
Types of Lights

- OpenGL supports two types of Lights
  - Local (Point) light sources
  - Infinite (Directional) light sources

- Type of light controlled by $w$ coordinate
  
  \[
  w = 0 \quad \text{Infinite Light directed along} \quad (x, y, z)
  \]
  
  \[
  w \neq 0 \quad \text{Local Light positioned at} \quad \left(\frac{x}{w}, \frac{y}{w}, \frac{z}{w}\right)
  \]
Turning on the Lights

- Flip each light’s switch
  ```c
  glEnable( GL_LIGHTn );
  ```
- Turn on the power
  ```c
  glEnable( GL_LIGHTING );
  ```
Controlling a Light’s Position

- Modelview matrix affects a light’s position
  - Different effects based on when position is specified
    - eye coordinates
    - world coordinates
    - model coordinates
  - Push and pop matrices to uniquely control a light’s position

Light Position Tutorial
Advanced Lighting Features

- **Spotlights**
  - localize lighting affects
    - `GL_SPOT_DIRECTION`
    - `GL_SPOT_CUTOFF`
    - `GL_SPOT_EXPONENT`

- **Light attenuation**
  - decrease light intensity with distance
    - `GL_CONSTANT_ATTENUATION`
    - `GL_LINEAR_ATTENUATION`
    - `GL_QUADRATIC_ATTENUATION`

\[ f_i = \frac{1}{k_c + k_l d + k_q d^2} \]
Light Model Properties

```c
glLightModelfv( property, value );
```

- Enabling two sided lighting
  ```
  GL_LIGHT_MODEL_TWO_SIDE
  ```
- Global ambient color
  ```
  GL_LIGHT_MODEL_AMBIENT
  ```
- Local viewer mode
  ```
  GL_LIGHT_MODEL_LOCAL_VIEWER
  ```
- Separate specular color
  ```
  GL_LIGHT_MODEL_COLOR_CONTROL
  ```
Tips for Better Lighting

- Recall lighting computed only at vertices
  - model tessellation heavily affects lighting results
    - better results but more geometry to process

- Use a single infinite light for fastest lighting
  - minimal computation per vertex
IMAGING AND RASTER PRIMITIVES
Imaging and Raster Primitives

- Describe OpenGL’s raster primitives: bitmaps and image rectangles
- Demonstrate how to get OpenGL to read and render pixel rectangles
Pixel-based primitives

- **Bitmaps**
  - 2D array of bit masks for pixels
    - update pixel color based on current color

- **Images**
  - 2D array of pixel color information
    - complete color information for each pixel

- **OpenGL doesn’t understand image formats**
Pixel Pipeline

- Programmable pixel storage and transfer operations

glBitmap(), glDrawPixels()

CPU \rightarrow Pixel Storage Modes \rightarrow Pixel-Transfer Operations (and Pixel Map) \rightarrow Rasterization (including Pixel Zoom) \rightarrow Per Fragment Operations \rightarrow Frame Buffer

Texture Memory

glCopyTexImage();

glReadPixels(), glCopyPixels()
Positioning Image Primitives

`glRasterPos3f(x, y, z)`

- raster position transformed like geometry
- discarded if raster position is outside of viewport
  - may need to fine tune viewport for desired results
Rendering Bitmaps

`glBitmap( width, height, xorig, yorig, xmove, ymove, bitmap )`

- render bitmap in current color at \((x - xorig, y - yorig)\)
- advance raster position by after rendering \((xmove, ymove)\)
OpenGL uses bitmaps for font rendering
  - each character is stored in a display list containing a bitmap
  - window system specific routines to access system fonts
    - glXUseXFont()
    - wglUseFontBitmaps()
Rendering Images

`glDrawPixels(width, height, format, type, pixels)`

- render pixels with lower left of image at current raster position
- numerous formats and data types for specifying storage in memory
  - best performance by using format and type that matches hardware
Reading Pixels

`glReadPixels(x, y, width, height, format, type, pixel)`

- read pixels from specified `(x, y)` position in framebuffer
- pixels automatically converted from framebuffer format into requested format and type

- Framebuffer pixel copy

  `glCopyPixels(x, y, width, height, type)`
**Pixel Zoom**

```c
glPixelZoom(x, y);
```

- expand, shrink or reflect pixels
- around current raster position
- fractional zoom supported

```c
Raster Position
```
Storage and Transfer Modes

- **Storage modes control accessing memory**
  - byte alignment in host memory
  - extracting a subimage

- **Transfer modes allow modify pixel values**
  - scale and bias pixel component values
  - replace colors using pixel maps

2. **Intro to 3D Graphics using Tao. OpenGL**, Erika Troll, Josh Lavinder, Alton Ng, Andrew Padilla
   - [http://www.math.ucla.edu/~wittman/10c.1.11s/Lectures/Raids/Graphics3D.pdf](http://www.math.ucla.edu/~wittman/10c.1.11s/Lectures/Raids/Graphics3D.pdf)

3. **OpenGL Programming Guide** *(Addison-Wesley Publishing Company)*, Denis Roegel, Lorraine Laboratory of IT Research and its Applications *(LORIA)*

4. **Learning Modern 3D Graphics Programming**, Jason L. McKesson

5. **An Interactive Introduction to OpenGL Programming**, Dave Shreiner, Ed Angel, Vicki Shreiner
   - [http://www.vis.uky.edu/~ryang/teaching/cs535-2012spr/Lectures/OpenGL.pptx, Lectures](http://www.vis.uky.edu/~ryang/teaching/cs535-2012spr/Lectures/OpenGL.pptx, Lectures)
Realizarea unei aplicatii simple care sa utilizeze:

1. Rendering Primitives
2. Rendering Modes
3. Lighting