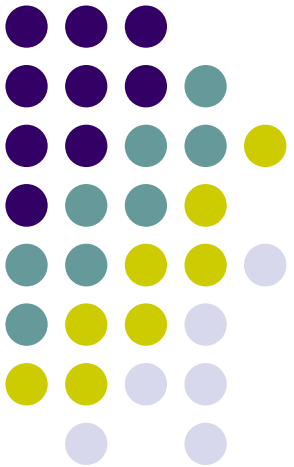
Computer Graphics
CS 543 – Lecture 4 (Part 2)
Building 3D Models (Part 2)

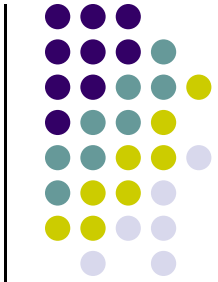
Prof Emmanuel Agu

Computer Science Dept.

Worcester Polytechnic Institute (WPI)



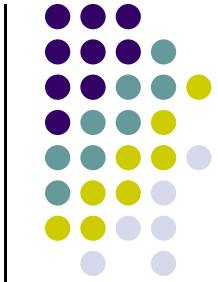
Modeling a Cube



- In 3D, declare vertices as (x,y,z) using **`point3 v[3]`**
- Define global arrays for vertices and colors

```
typedef vex3 point3;
point3 vertices[] = {point3(-1.0,-1.0,-1.0),
  point3(1.0,-1.0,-1.0), point3(1.0,1.0,-1.0),
  point3(-1.0,1.0,-1.0), point3(-1.0,-1.0,1.0),
  point3(1.0,-1.0,1.0), point3(1.0,1.0,1.0),
  point3(-1.0,1.0,1.0)};

    typedef vec3 color3;
    color3 colors[] = {color3(0.0,0.0,0.0),
      color3(1.0,0.0,0.0), color3(1.0,1.0,0.0),
      color(0.0,1.0,0.0), color3(0.0,0.0,1.0),
      color3(1.0,0.0,1.0), color3(1.0,1.0,1.0),
      color3(0.0,1.0,1.0});
```

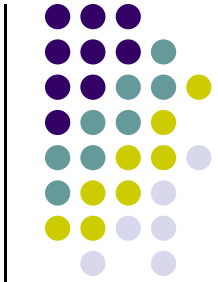
Drawing a triangle from list of indices



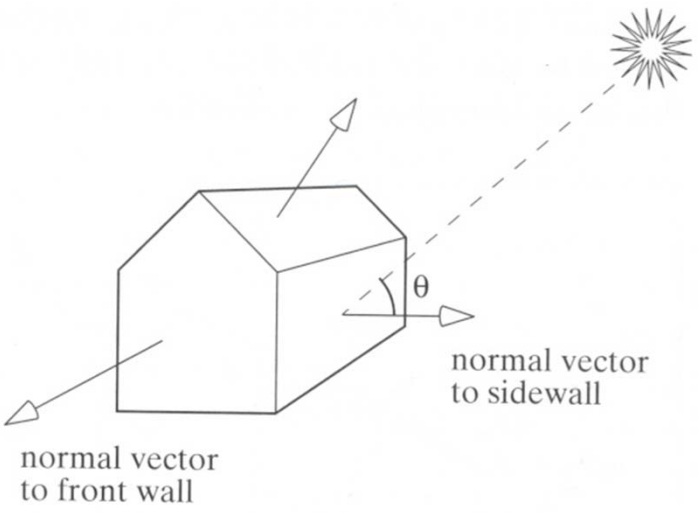
Draw a triangle from a list of indices into the array **`vertices`** and assign a color to each index

```
void triangle(int a, int b, int c, int d)
{
   vcolors[i] = colors[d];
   position[i] = vertices[a];
   vcolors[i+1] = colors[d]);
   position[i+1] = vertices[b];
   vcolors[i+2] = colors[d];
   position[i+2] = vertices[c];
   i+=3;
 }
```

Normal Vector

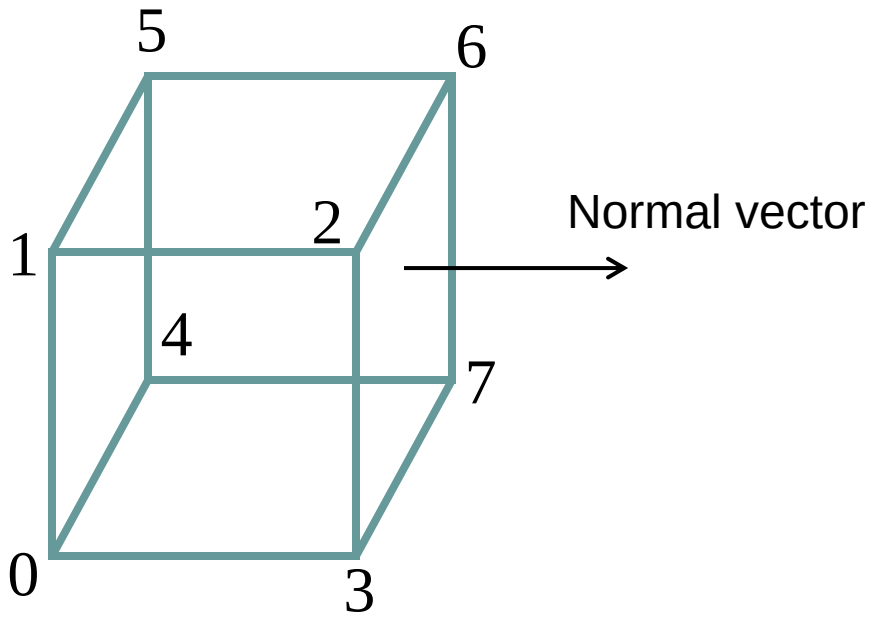


- **Normal vector:** Direction of each polygon
- Each mesh polygon has a **normal vector**
- Normal vector used in shading
- **Normal vector • light vector** determines shading



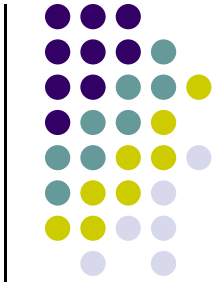
Draw cube from faces

```
void colorcube( )
{
    quad(0,3,2,1);
    quad(2,3,7,6);
    quad(0,4,7,3);
    quad(1,2,6,5);
    quad(4,5,6,7);
    quad(0,1,5,4);
}
```



Note: vertices ordered so that we obtain correct outward facing normals

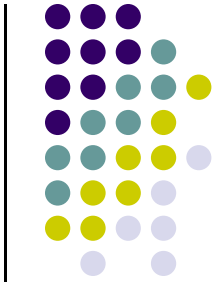
Old Way: Inefficient



- Previously drew cube by its 6 faces using
  - 6 **glBegin**, 6 **glEnd**
  - 6 **glColor**
  - 24 **glVertex**
  - More commands if we use texture and lighting
  - E.g: to draw each face

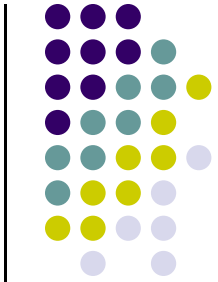
```
glBegin(GL_QUAD)
        glVertex(x1, y1, z1);
        glVertex(x2, y2, z2);
        glVertex(x3, y3, z3);
        glVertex(x4, y4, z4);
glEnd( );
```

Vertex Arrays



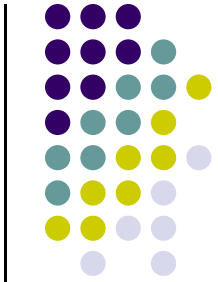
- **Previously:** OpenGL provided a facility called ***vertex arrays*** for storing rendering data
- Six types of arrays were supported initially
  - Vertices
  - Colors
  - Color indices
  - Normals
  - Texture coordinates
  - Edge flags
- Now vertex arrays can be used for any attributes

New Way: Drawing the cube



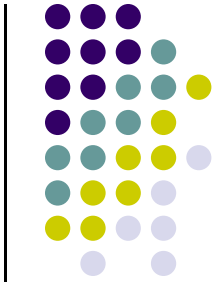
- Drawing Similar to 2D
  - Move array of 3D mesh vertices to **vertex buffer object**
  - Draw mesh using **glDrawArrays**

Full Example: Rotating Cube



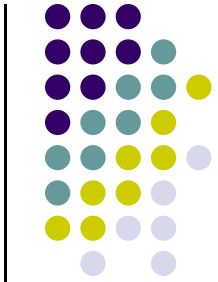
- Program behaviour:
  - Draw colored cube
  - Use 3-button mouse to change direction of rotation
  - Use idle function to increment angle of rotation

Cube Vertices



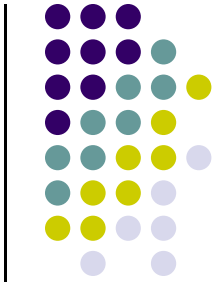
```
// Vertices of a unit cube centered at origin
//  sides aligned with axes

point4 vertices[8] = {
    point4( -0.5, -0.5,  0.5, 1.0 ),
    point4( -0.5,  0.5,  0.5, 1.0 ),
    point4(  0.5,  0.5,  0.5, 1.0 ),
    point4(  0.5, -0.5,  0.5, 1.0 ),
    point4( -0.5, -0.5, -0.5, 1.0 ),
    point4( -0.5,  0.5, -0.5, 1.0 ),
    point4(  0.5,  0.5, -0.5, 1.0 ),
    point4(  0.5, -0.5, -0.5, 1.0 )
};
```

Colors



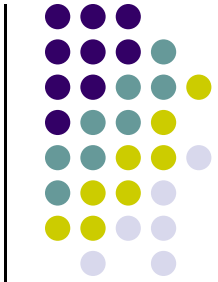
```
// RGBA colors

color4 vertex_colors[8] = {
    color4( 0.0, 0.0, 0.0, 1.0 ),  // black
    color4( 1.0, 0.0, 0.0, 1.0 ),  // red
    color4( 1.0, 1.0, 0.0, 1.0 ),  // yellow
    color4( 0.0, 1.0, 0.0, 1.0 ),  // green
    color4( 0.0, 0.0, 1.0, 1.0 ),  // blue
    color4( 1.0, 0.0, 1.0, 1.0 ),  // magenta
    color4( 1.0, 1.0, 1.0, 1.0 ),  // white
    color4( 0.0, 1.0, 1.0, 1.0 )   // cyan
};
```

Quad Function



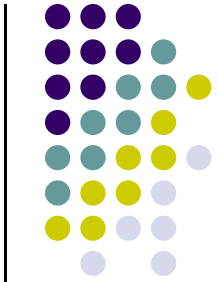
```
// quad generates two triangles (a,b,c) and (a,c,d) for each face and
// assigns colors to the vertices

int Index = 0;  // Index goes from 1 to 6, one per face

void quad( int a, int b, int c, int d )
{
    colors[Index] = vertex_colors[a]; points[Index] = vertices[a]; Index++
    colors[Index] = vertex_colors[b]; points[Index] = vertices[b]; Index++
    colors[Index] = vertex_colors[c]; points[Index] = vertices[c]; Index++
    colors[Index] = vertex_colors[a]; points[Index] = vertices[a]; Index++
    colors[Index] = vertex_colors[c]; points[Index] = vertices[c]; Index++
    colors[Index] = vertex_colors[d]; points[Index] = vertices[d]; Index++
}
```

Color Cube



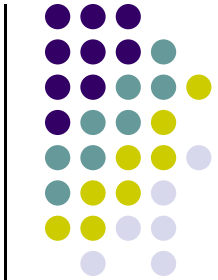
```
// generate 12 triangles: 36 vertices and 36 colors

void colorcube()
{
    quad( 1, 0, 3, 2 );
    quad( 2, 3, 7, 6 );
    quad( 3, 0, 4, 7 );
    quad( 6, 5, 1, 2 );
    quad( 4, 5, 6, 7 );
    quad( 5, 4, 0, 1 );
}
```

Initialization I



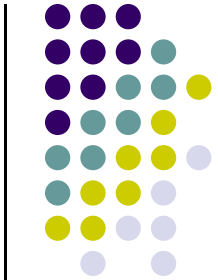
```
void init()
{
    colorcube(); // Generates cube data in application


    // Create a vertex array object

    GLuint vao;
    glGenVertexArrays ( 1, &vao );
    glBindVertexArray ( vao );
```

Initialization II



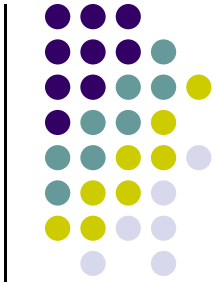
```
// Create and initialize a buffer object and move points
// data to GPU

    GLuint buffer;
    glGenBuffers( 1, &buffer );
    glBindBuffer( GL_ARRAY_BUFFER, buffer );
    glBufferData( GL_ARRAY_BUFFER, sizeof(points) +
                  sizeof(colors), NULL, GL_STATIC_DRAW );
```

Initialization III



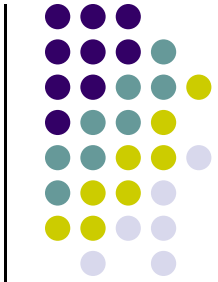
```
    glBufferSubData( GL_ARRAY_BUFFER, 0,
                                 sizeof(points), points );
    glBufferSubData( GL_ARRAY_BUFFER, sizeof(points),
                                 sizeof(colors), colors );

// Load shaders and use the resulting shader program
    GLuint program = InitShader( "vshader36.glsl",
                                          "fshader36.glsl" );
    glUseProgram( program );
```

Initialization IV



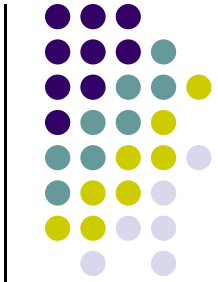
```
// set up vertex arrays

    GLuint vPosition = glGetAttribLocation( program, "vPosition" );
    glEnableVertexAttribArray( vPosition );
    glVertexAttribPointer( vPosition, 4, GL_FLOAT, GL_FALSE, 0,
                           BUFFER_OFFSET(0) );

    GLuint vColor = glGetAttribLocation( program, "vColor" );
    glEnableVertexAttribArray( vColor );
    glVertexAttribPointer( vColor, 4, GL_FLOAT, GL_FALSE, 0,
                           BUFFER_OFFSET(sizeof(points)) );

    theta = glGetUniformLocation( program, "theta" );
```

Display Callback



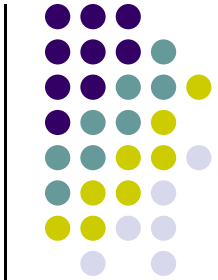
```
void display( void )
{
    glClear( GL_COLOR_BUFFER_BIT
                            |GL_DEPTH_BUFFER_BIT );

    glUniform3fv( theta, 1, theta );
    glDrawArrays( GL_TRIANGLES, 0, NumVertices );

    glutSwapBuffers();
}
```

Mouse Callback



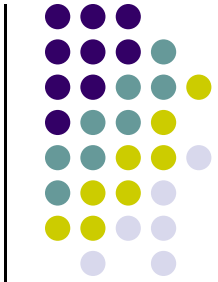
```
void mouse( int button, int state, int x, int y )
{
    if ( state == GLUT_DOWN ) {
        switch( button ) {
            case GLUT_LEFT_BUTTON:    axis = Xaxis;  break;
            case GLUT_MIDDLE_BUTTON:  axis = Yaxis;  break;
            case GLUT_RIGHT_BUTTON:   axis = Zaxis;  break;
        }
    }
}
```

Idle Callback

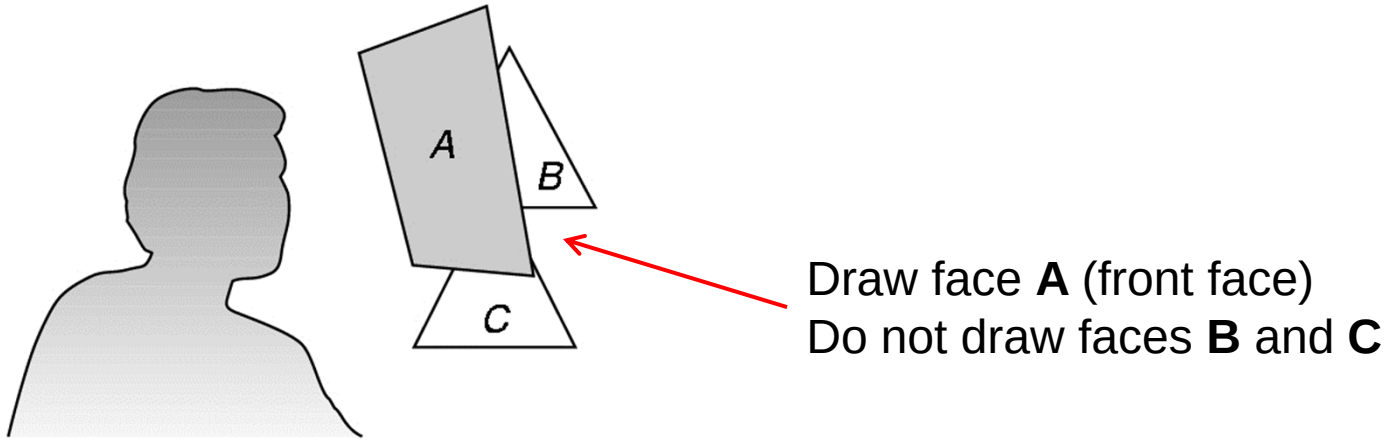


```
void idle( void )
{
    theta[axis] += 0.01;

    if ( theta[axis] > 360.0 ) {
        theta[axis] -= 360.0;
    }

    glutPostRedisplay();
}
```

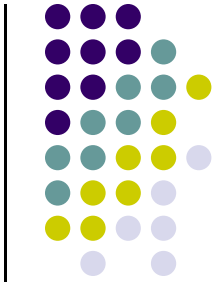
Hidden-Surface Removal



- We want to see only surfaces in front of other surfaces
- OpenGL uses *hidden-surface* technique called the ***z-buffer*** algorithm
- Z-buffer uses distance from viewer (depth) to determine closer objects
- Objects rendered so that only front objects appear in image



Using OpenGL's *z*-buffer algorithm



- Z-buffer uses an extra buffer, (the z-buffer), to store depth information as geometry travels down the pipeline
- 3 steps to set up Z-buffer:
  1. Requested in **`main.c`**

     ```
     glutInitDisplayMode
       (GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH)
     ```

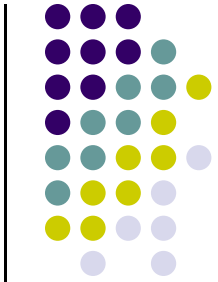
  2. Enabled in **`init.c`**

     ```
     glEnable(GL_DEPTH_TEST)
     ```

  3. Cleared in the display callback

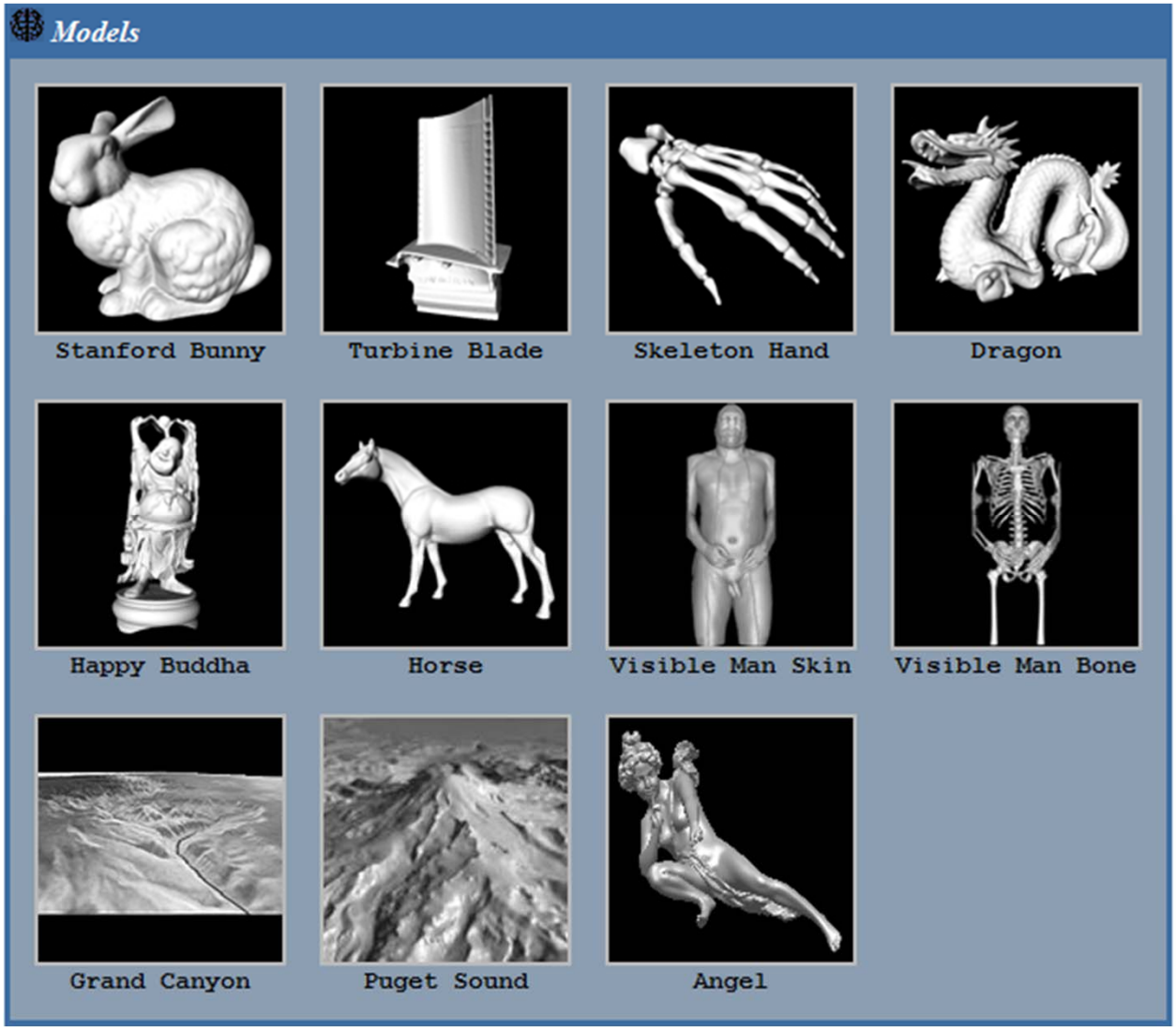
     ```
     glClear(GL_COLOR_BUFFER_BIT |
                              GL_DEPTH_BUFFER_BIT)
     ```

3D Mesh file formats



- 3D meshes usually stored in 3D file format
- Format defines how vertices, edges, and faces are declared
- Over 400 different file format
- **Polygon File Format (PLY)** used a lot in graphics
- Originally PLY was used to store 3D files from 3D scanner
- We can get PLY models from web to work with
- We will use PLY files in this class

Georgia Tech Large Models Archive

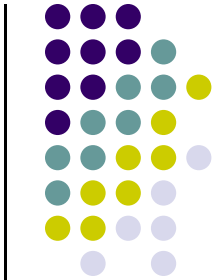


Stanford 3D Scanning Repository

Lucy: 28 million faces

Happy Buddha: 9 million faces

References



- Angel and Shreiner
- Hill and Kelley