New Way: Vertex Representation and Storage

- We have declared vertex lists, edge lists, and arrays.
- But vertex data is usually passed to OpenGL in an array with a specific structure.
- We now study that structure....
Vertices can have attributes

- Position (e.g. 20, 12, 18)
- Color (e.g. red)
- Normal (x,y,z)
- Texture coordinates
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
Vertex Attributes

- Store vertex attributes in **single** Array (array of structures)
Declaring Array of Vertex Attributes

- Consider the following array of vertex attributes

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
<th>r</th>
<th>g</th>
<th>b</th>
<th>s</th>
<th>t</th>
<th>s</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position</td>
<td>Color</td>
<td>Tex0</td>
<td>Tex1</td>
<td>Tex0</td>
<td>Tex1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- So we can define attribute positions (per vertex)

```c
#define VERTEX_POS_INDEX 0
#define VERTEX_COLOR_INDEX 1
#define VERTEX_TEXCOORD0_INDEX 2
#define VERTEX_TEXCOORD1_INDEX 3
```
Declaring Array of Vertex Attributes

- Also define number of floats (storage) for each vertex attribute

```c
#define VERTEX_POS_SIZE 3  // x, y and z
#define VERTEX_COLOR_SIZE 3  // r, g and b
#define VERTEX_TEXCOORD0_SIZE 2  // s and t
#define VERTEX_TEXCOORD1_SIZE 2  // s and t

#define VERTEX_ATTRIB_SIZE VERTEX_POS_SIZE + VERTEX_COLOR_SIZE + VERTEX_TEXCOORD0_SIZE + VERTEX_TEXCOORD1_SIZE
```
Declaring Array of Vertex Attributes

- Define offsets (# of floats) of each vertex attribute from beginning

```
#define VERTEX_POS_OFFSET 0
#define VERTEX_COLOR_OFFSET 3
#define VERTEX_TEXCOORD0_OFFSET 6
#define VERTEX_TEXCOORD1_OFFSET 8
```
Allocating Array of Vertex Attributes

- Allocate memory for entire array of vertex attributes

```
#define VERTEX_ATTRIB_SIZE VERTEX_POS_SIZE + VERTEX_COLOR_SIZE + \
                        VERTEX_TEXCOORD0_SIZE + \
                        VERTEX_TEXCOORD1_SIZE

float *p = malloc(numVertices * VERTEX_ATTRIB_SIZE * sizeof(float));
```

Allocate memory for all vertices
### Specifying Array of Vertex Attributes

- **glVertexAttribPointer** used to specify vertex attributes
- Example: to specify vertex position attribute

```c
void glVertexAttribPointer(GLuint index, GLuint size, GLenum type, GLboolean normalize, GLsizei stride, const void *pointer);

void glEnableVertexAttribArray(GLuint index);
```

<table>
<thead>
<tr>
<th>Position</th>
<th>Color</th>
<th>Tex0</th>
<th>Tex1</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>y</td>
<td>z</td>
<td></td>
</tr>
<tr>
<td>r</td>
<td>g</td>
<td>b</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Position</th>
<th>Color</th>
<th>Tex0</th>
<th>Tex1</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>y</td>
<td>z</td>
<td></td>
</tr>
<tr>
<td>r</td>
<td>g</td>
<td>b</td>
<td></td>
</tr>
</tbody>
</table>

- Data is floats
- Stride: distance between consecutive vertices
- Pointer to data
- Data should not be normalized

```
glVertexAttribPointer(VERTEX_POS_INDEX, VERTEX_POS_SIZE, GL_FLOAT, GL_FALSE, VERTEX_ATTRIB_SIZE * sizeof(float), p);
glEnableVertexAttribArray(0);
```

- do same for normal, tex0 and tex1
Computer Graphics (CS 4731)  
Lecture 8: Building 3D Models & Introduction to Transformations  

Prof Emmanuel Agu  

Computer Science Dept.  
Worcester Polytechnic Institute (WPI)
Full Example: Rotating Cube in 3D

- **Desired Program behaviour:**
  - Draw colored cube
  - Continuous rotation about X, Y or Z axis
    - Idle function called repeatedly when nothing to do
    - Increment angle of rotation in idle function
  - Use 3-button mouse to change direction of rotation
    - Click left button → rotate cube around X axis
    - Click middle button → rotate cube around Y axis
    - Click right button → rotate cube around Z axis

- **Use default camera**
  - If we don’t set camera, we get a default camera
  - Located at origin and points in the negative z direction
Cube Vertices

Declare array of \((x,y,z,w)\) vertex positions for a unit cube centered at origin (Sides aligned with axes)

```cpp
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 )
};
```

Declare array of vertex colors (set of RGBA colors vertex can have)

```cpp
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
};
```
Color Cube

// generate 6 quads,
// sides of cube

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 );
}

Function quad is
Passed vertex indices

point4 vertices[8] = {
    0 point4( -0.5, -0.5,  0.5, 1.0 ),
    1 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 ),
    4 point4( -0.5, -0.5, -0.5, 1.0 ),
    5 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 )
};
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 0 to 5, one for each vertex of 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++;
}

quad 0 = points[0 - 5 ]
quad 1 = points[6 - 11]
quad 2 = points [12 - 17] ...etc

Points[ ] array to be
Sent to GPU

Read from appropriate index
of unique positions declared
void init()
{
    colorcube(); // Generates cube data in application using quads

    // Create a vertex array object
    GLuint vao;
    glGenVertexArrays ( 1, &vao );
    glBindVertexArray ( vao );

    // Create a buffer object and move 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 );
}

Points[ ] array of vertex positions sent to GPU
colors[ ] array of vertex colors sent to GPU
Initialization II

Send *points[*] and *colors[*] data to GPU separately using `glBufferSubData`

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

// Load vertex and fragment shaders and use the resulting shader program
GLuint program = InitShader( "vshader36.glsl", "fshader36.glsl" );
glUseProgram( program );
// set up vertex arrays

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

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

theta = glGetUniformLocation( program, "theta" );

Want to Connect rotation variable theta in program to variable in shader
void display( void )
{
    glClear( GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT );

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

    glutSwapBuffers();
}
enum { Xaxis = 0, Yaxis = 1, Zaxis = 2, NumAxes = 3 };
Idle Callback

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

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

    glutPostRedisplay();
}

void main( void )
{
    ........

    glutIdleFunc( idle );
    ........
}

The idle( ) function is called whenever nothing to do

Use it to increment rotation angle in steps of theta = 0.01 around currently selected axis

Note: still need to:
• Apply rotation by (theta) in shader
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

If overlap,
- Draw face A (front face)
- Do not draw faces B and C
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. In `main()` function
     ```
     glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH)
     ```
  2. Enabled in `init()` function
     ```
     glEnable(GL_DEPTH_TEST)
     ```
  3. Clear depth buffer whenever we clear screen
     ```
     glClear(GL_COLOR_BUFFER_BIT | 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 formats
- **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
Sample PLY File

ply
format ascii 1.0
comment this is a simple file
obj_info any data, in one line of free form text
element vertex 3
property float x
property float y
property float z
element face 1
property list uchar int vertex_indices
end_header
-1 0 0
0 1 0
1 0 0
3 0 1 2
Georgia Tech Large Models Archive

- Stanford Bunny
- Turbine Blade
- Skeleton Hand
- Dragon
- Happy Buddha
- Horse
- Visible Man Skin
- Visible Man Bone
- Grand Canyon
- Puget Sound
- Angel
Stanford 3D Scanning Repository

Lucy: 28 million faces

Happy Buddha: 9 million faces
Introduction to Transformations

- May also want to transform objects by changing its:
  - Position (translation)
  - Size (scaling)
  - Orientation (rotation)
  - Shapes (shear)
Translation

- Move each vertex by **same** distance \( \mathbf{d} = (d_x, d_y, d_z) \)

object

translation: every point displaced by same vector
Scaling

Expand or contract along each axis (fixed point of origin)

\[ x' = s_x x \]
\[ y' = s_y y \]
\[ z' = s_z z \]

\[ p' = Sp \]

where

\[ S = S(s_x, s_y, s_z) \]