Lab 1
Vertex Array

CSE 4431/5331.03M

Advanced Topics in 3D Computer Graphics

TA: Margarita Vinnikov

mvinni@cse.yorku.ca

Installing GLUT

At home

- The official site for GLUT for Windows is:

http://www.xmission.com/~nate/glut.html

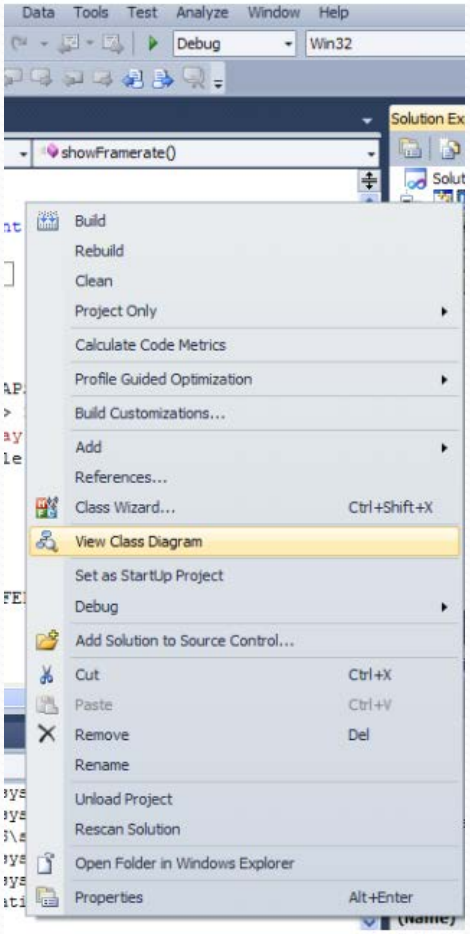
- On your home machine
  - Place the files in the directories as indicated
    - glut32.dll -> C:\Windows\System or C:\WinNT\System
    - glut32.lib -> C:\Program Files\Microsoft Visual Studio 8\VC\PlatformSDK\lib
    - glut32.h -> C:\Program Files\Microsoft Visual Studio8\VC\PlatformSDK\include\gl

- Note: In order for someone else on a different machine to run your application you must include the glut32.dll file with the program.

- Other way to install and other issues with glut:
  - http://www.lighthouse3d.com/opengl/glut/

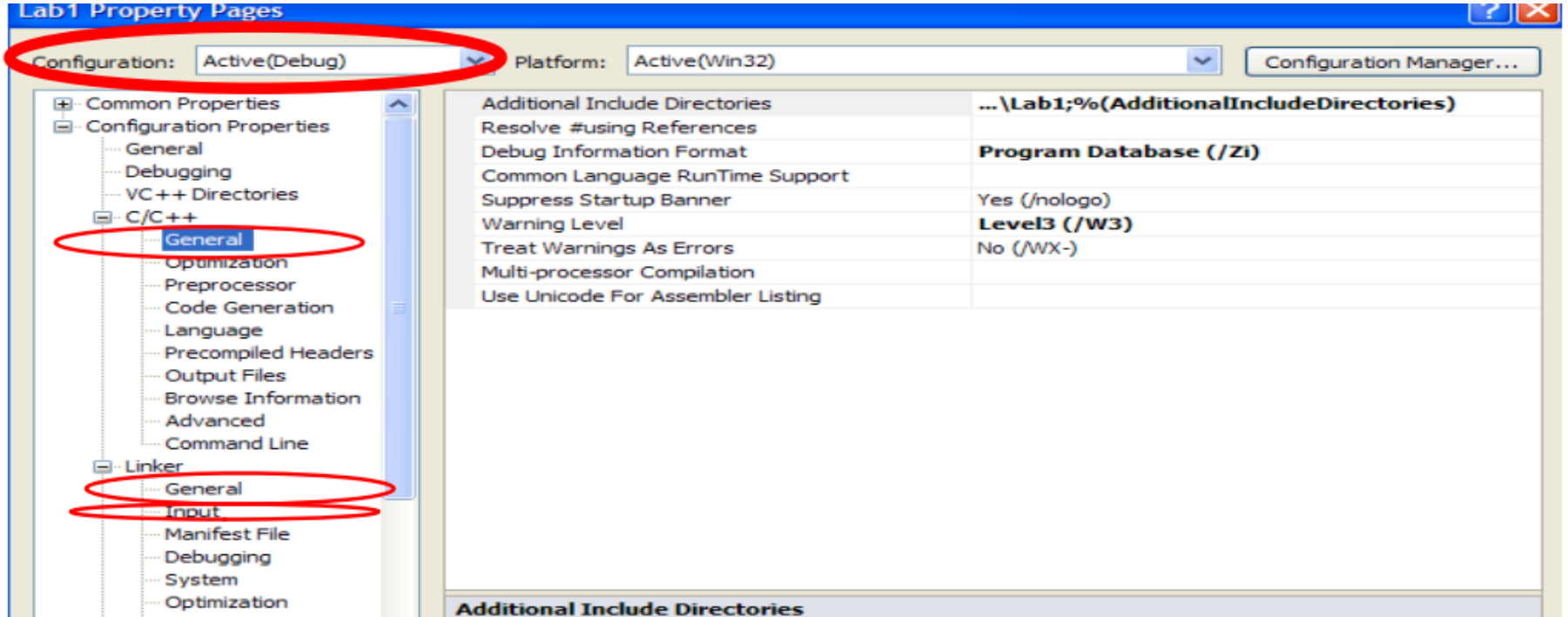
In the lab (1)

1. Create project Labo1 in Microsoft Visual Studio
2. Download glut library from http://www.xmission.com/~nate/glut.html
3. Placed the files in your project directory



In the lab (2)

4. Change the project's properties:
   - Configuration → All configurations
   - C/C++ → General → add additional directory → add your directory with glut library
   - Linker → General → add additional directory → add your directory with glut library
   - Linker → Input → add lib → glut32.lib

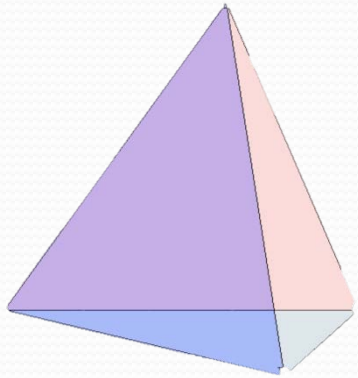


First task

Draw Tetrahedron

- Use the template code provided with the lab.
- Explore the code in the provided template.
- In the initScene() add the following:

```
glCullFace(GL_BACK);
glEnable(GL_CULL_FACE);
```



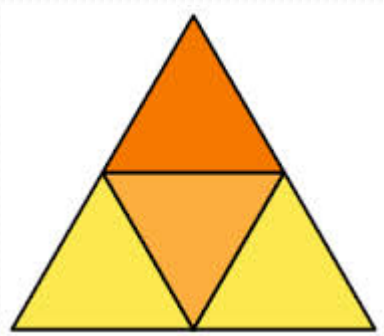
- Make a procedure that draws a Tetrahedron :
  - Example the Cartesian coordinates of the vertices are

(+0.2, +0.2, +0.2)

(−0.2, −0.2, +0.2)

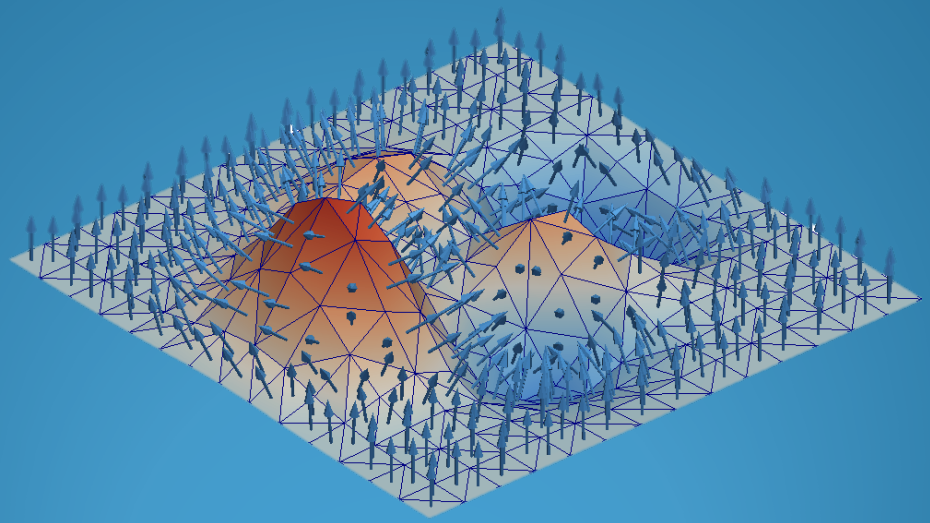
(−0.2, +0.2, −0.2)

(+0.2, −0.2, −0.2)



- Associate color for each vertex, color your model

Normals



Culling

- Culling calculates the signed area of the filled primitive in window coordinate space.
- The signed area is
  - positive when the window coordinates are in a counter-clockwise order
  - negative when clockwise
- Use glFrontFace() to specify the ordering, counter-clockwise or clockwise, to be interpreted as a front-facing or back-facing primitive.
- Specify culling either front or back faces by calling glCullFace().
- Enable with a call to glEnable(GL_CULL_FACE);

Normal Vector

- A normal is a vector that points in a direction that is perpendicular to a surface
- Can be defined per
  - Face
  - Vertex
- If the application has performed non-uniform scaling the normals will no longer be correct.
  - Call `glEnable(GL_NORMALIZE)`

Per Face

- Can be defined per
  - Face
    1. Find two vectors coplanar with the face.

       v1 = t2 - t1 v2 = t3 - t1
    2. Compute the cross product will provide the normal vector.

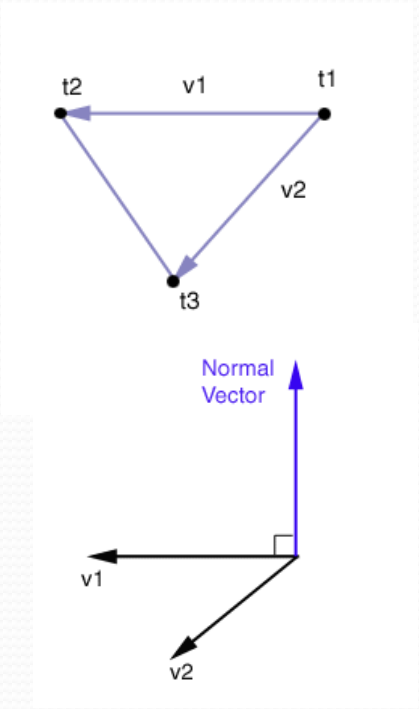
    vx = v1y * v2z - v1z * v2y

    vy = v1z * v2x - v1x * v2z

    vz = v1x * v2y - v1y * v2x
    3. Normalise (make it unit length)
       1. Compute the length

          L = sqrt(vx*vx + vy*vy + vz*vz)
       2. Dividing each component by the vectors length.

       nvx = vx / l nvy = vy / l nvz = vz / l

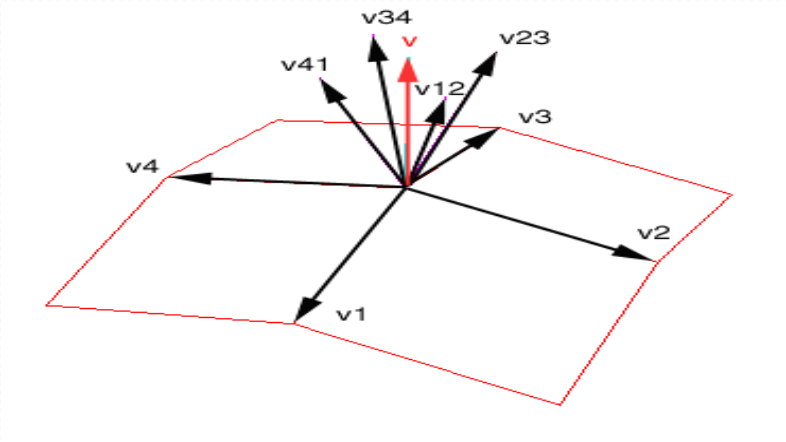


http://www.lighthouse3d.com/opengl/terrain/index.php3?normals

Per Vertex

- Smoother look
- Consider the faces that share the vertex.
  - Should be computed as :
    - the normalised sum of all the unit length normals for each face the vertex shares.
    - Example :

```
v = normalised(sum(v12, v23, v34, v41)) where
vij = normalised(vi x vj) // normalised cross product
```



http://www.lighthouse3d.com/opengl/terrain/index.php3?normals

Implementation Example

```
void normalizesLength(float vector[3]) // Reduces A Normal Vector (3 Coordinates)
{
  length = (float)sqrt((vector[0]*vector[0])+(vector[1]*vector[1])+vector[2]*vector[2]));
    if(length == 0.0f) // Prevents Divide By 0 Error By Providing
         length = 1.0f; // An Acceptable Value For Vectors To Close To 0.
    vector[0] /= length; // Dividing Each Element By
    vector[1] /= length; // The Length Results In A
    vector[2] /= length; // Unit Normal Vector.
 }
void calcNormal(float v[3][3], float out[3]) // Calculates Normal For A face Using 3 Points
 {
    float v1[3],v2[3]; // Vector 1 (x,y,z) & Vector 2 (x,y,z)
    static const int x = 0; static const int y = 1; static const int z = 2;
    // Finds The Vector Between 2 Points By Subtracting the x,y,z Coordinates
    v1[x] = v[0][x] - v[1][x]; v1[y] = v[0][y] - v[1][y]; v1[z] = v[0][z] - v[1][z];
    //Calculate The Vector From Point 2 To Point 1
    v2[x] = v[1][x] - v[2][x]; v2[y] = v[1][y] - v[2][y]; v2[z] = v[1][z] - v[2][z];
    Compute The Cross Product To Give Us A Surface Normal
    out[x] = v1[y]*v2[z] - v1[z]*v2[y]; // Cross Product For Y - Z
    out[y] = v1[z]*v2[x] - v1[x]*v2[z]; // Cross Product For X - Z
    out[z] = v1[x]*v2[y] - v1[y]*v2[x]; // Cross Product For X - Y
    normalizesLength(out); // Normalize The Vectors
}
```

Second task

Normals

- Calculate normals to each face

Bonus:

- In the updateScene() add code to change the location of your figure's vertices so the figure will deform as the time will pass. Your will have to recalculate normals every time you make a change.

Vertex Array

Vertex Array Prep.

- Count
  - How many glVertex*() calls did you make?
  - How many glColor*() calls did you make?
  - How many normals to the corresponding vertices did you add?
  - How many times did you repeat a call to the same vertex?
- What is current frame rate performance?

- Why?
  - Reduces the number of function calls and redundant usage of shared vertices.
  - Improves performance.

Example

```
//    v6----- v5
//   /|      /|
//  v1------v0|
//  | |     | |
//  | |v7---|-|v4
//  |/      |/
//  v2------v3
```

Declaration

```
// vertex coords array
GLfloat vertices[] = {1,1,1,  -1,1,1,  -1,-1,1,  1,-1,1,        // v0-v1-v2-v3
                      1,1,1,  1,-1,1,  1,-1,-1,  1,1,-1,        // v0-v3-v4-v5
                      1,1,1,  1,1,-1,  -1,1,-1,  -1,1,1,        // v0-v5-v6-v1
                      -1,1,1,  -1,1,-1,  -1,-1,-1,  -1,-1,1,    // v1-v6-v7-v2
                      -1,-1,-1,  1,-1,-1,  1,-1,1,  -1,-1,1,    // v7-v4-v3-v2
                      1,-1,-1,  -1,-1,-1,  -1,1,-1,  1,1,-1};   // v4-v7-v6-v5

// normal array
GLfloat normals[] = {0,0,1,  0,0,1,  0,0,1,  0,0,1,           // v0-v1-v2-v3
                     1,0,0,  1,0,0,  1,0,0, 1,0,0,            // v0-v3-v4-v5
                     0,1,0,  0,1,0,  0,1,0, 0,1,0,            // v0-v5-v6-v1
                     -1,0,0,  -1,0,0, -1,0,0,  -1,0,0,        // v1-v6-v7-v2
                     0,-1,0,  0,-1,0,  0,-1,0,  0,-1,0,       // v7-v4-v3-v2
                     0,0,-1,  0,0,-1,  0,0,-1,  0,0,-1};      // v4-v7-v6-v5

// color array
GLfloat colors[] = {1,1,1,  1,1,0,  1,0,0,  1,0,1,           // v0-v1-v2-v3
                    1,1,1,  1,0,1,  0,0,1,  0,1,1,           // v0-v3-v4-v5
                    1,1,1,  0,1,1,  0,1,0,  1,1,0,           // v0-v5-v6-v1
                    1,1,0,  0,1,0,  0,0,0,  1,0,0,           // v1-v6-v7-v2
                    0,0,0,  0,0,1,  1,0,1,  1,0,0,           // v7-v4-v3-v2
                    0,0,1,  0,0,0,  0,1,0,  0,1,1};          // v4-v7-v6-v5
```

Initialization

```
Void drawFunction()
{
   // enble and specify pointers to vertex arrays
   glEnableClientState(GL_NORMAL_ARRAY);
   glEnableClientState(GL_COLOR_ARRAY);
   glEnableClientState(GL_VERTEX_ARRAY);

   glNormalPointer(GL_FLOAT, 0, normals);
   glColorPointer(3, GL_FLOAT, 0, colors);
   glVertexPointer(3, GL_FLOAT, 0, vertices);

   glDrawArrays(GL_QUADS, 0, 24);

   glDisableClientState(GL_VERTEX_ARRAY);
   glDisableClientState(GL_COLOR_ARRAY);
   glDisableClientState(GL_NORMAL_ARRAY);
}
```

Diactivation

Step 1 – Initialization I

1. Enabling Arrays
   - Functions to activate and deactivate 8 different types of arrays.
     - *void* ***glEnableClientState****(GLenum array)*
     - *void* ***glDisableClientState****(GLenum array);*
       - *Symbolic constants GL_VERTEX_ARRAY, GL_COLOR_ARRAY, GL_SECONDARY_COLOR_ARRAY,GL_INDEX_ARRAY, GL_NORMAL_ARRAY, GL_FOG_ARRAY, GL_TEXTURE_COORD_ARRAY, and GL_EDGE_FLAG_ARRAY*

Step 1 – Initialization II

2. Specifying data for the array
   - Specify the exact positions(addresses) of arrays,
     - **glVertexPointer()**: specify pointer to vertex coords array
     - **glNormalPointer()**: specify pointer to normal array
     - **glColorPointer()**: specify pointer to RGB color array
     - **glSecondaryColorPointer()**: specify pointer to secondary RGB color array
     - **glIndexPointer()**: specify pointer to indexed color array
     - **glFogCoordPointer():** specify pointer to indexed color array
     - **glTexCoordPointer()**: specify pointer to texture cords array
     - **glEdgeFlagPointer()**: specify pointer to edge flag array

Example

```
//    v6----- v5
//   /|      /|
//  v1------v0|
//  | |     | |
//  | |v7---|-|v4
//  |/      |/
//  v2------v3
```

```
// vertex coords array
GLfloat vertices[] = {1,1,1,  -1,1,1,  -1,-1,1,  1,-1,1,        // v0-v1-v2-v3
                      1,1,1,  1,-1,1,  1,-1,-1,  1,1,-1,        // v0-v3-v4-v5
                      1,1,1,  1,1,-1,  -1,1,-1,  -1,1,1,        // v0-v5-v6-v1
                      -1,1,1,  -1,1,-1,  -1,-1,-1,  -1,-1,1,    // v1-v6-v7-v2
                      -1,-1,-1,  1,-1,-1,  1,-1,1,  -1,-1,1,    // v7-v4-v3-v2
                      1,-1,-1,  -1,-1,-1,  -1,1,-1,  1,1,-1};   // v4-v7-v6-v5

// normal array
GLfloat normals[] = {0,0,1,  0,0,1,  0,0,1,  0,0,1,           // v0-v1-v2-v3
                     1,0,0,  1,0,0,  1,0,0, 1,0,0,            // v0-v3-v4-v5
                     0,1,0,  0,1,0,  0,1,0, 0,1,0,            // v0-v5-v6-v1
                     -1,0,0,  -1,0,0, -1,0,0,  -1,0,0,        // v1-v6-v7-v2
                     0,-1,0,  0,-1,0,  0,-1,0,  0,-1,0,       // v7-v4-v3-v2
                     0,0,-1,  0,0,-1,  0,0,-1,  0,0,-1};      // v4-v7-v6-v5

// color array
GLfloat colors[] = {1,1,1,  1,1,0,  1,0,0,  1,0,1,           // v0-v1-v2-v3
                    1,1,1,  1,0,1,  0,0,1,  0,1,1,           // v0-v3-v4-v5
                    1,1,1,  0,1,1,  0,1,0,  1,1,0,           // v0-v5-v6-v1
                    1,1,0,  0,1,0,  0,0,0,  1,0,0,           // v1-v6-v7-v2
                    0,0,0,  0,0,1,  1,0,1,  1,0,0,           // v7-v4-v3-v2
                    0,0,1,  0,0,0,  0,1,0,  0,1,1};          // v4-v7-v6-v5
```

```
Void drawFunction()
{
   // enble and specify pointers to vertex arrays
   glEnableClientState(GL_NORMAL_ARRAY);
   glEnableClientState(GL_COLOR_ARRAY);
   glEnableClientState(GL_VERTEX_ARRAY);

   glNormalPointer(GL_FLOAT, 0, normals);
   glColorPointer(3, GL_FLOAT, 0, colors);
   glVertexPointer(3, GL_FLOAT, 0, vertices);

   glDrawArrays(GL_QUADS, 0, 24);

   glDisableClientState(GL_VERTEX_ARRAY);
   glDisableClientState(GL_COLOR_ARRAY);
   glDisableClientState(GL_NORMAL_ARRAY);
}
```

Step 1 – Initialization II Specifics

- *void **glVertexPointer**(GLint size, GLenum type, GLsizei stride, const GLvoid *pointer);*
- *void **glColorPointer**(GLint size, GLenum type, GLsizei stride, const GLvoid *pointer);*
- *void **glIndexPointer**(GLenum type, GLsizei stride, const GLvoid *pointer);*
- *void **glNormalPointer**(GLenum type, GLsizei stride, const GLvoid *pointer);*
- *void **glTexCoordPointer**(GLint size, GLenum type, GLsizei stride, const GLvoid *pointer);*
- *void **glEdgeFlagPointer**(GLsizei stride, const GLvoid *pointer);*

Command	Sizes	Values for *type* Argument
glVertexPointer	2, 3, 4	GL_SHORT, GL_INT, GL_FLOAT, GL_DOUBLE
glNormalPointer	3	GL_BYTE, GL_SHORT, GL_INT, GL_FLOAT, GL_DOUBLE
glColorPointer	3, 4	GL_BYTE, GL_UNSIGNED_BYTE, GL_SHORT, GL_UNSIGNED_SHORT, GL_INT, GL_UNSIGNED_INT, GL_FLOAT, GL_DOUBLE
glSecondaryColorPointer	3	GL_BYTE, GL_UNSIGNED_BYTE, GL_SHORT, GL_UNSIGNED_SHORT, GL_INT, GL_UNSIGNED_INT, GL_FLOAT, GL_DOUBLE
glIndexPointer	1	GL_UNSIGNED_BYTE, GL_SHORT, GL_INT, GL_FLOAT, GL_DOUBLE
glTexCoordPointer	1, 2, 3, 4	GL_SHORT, GL_INT, GL_FLOAT, GL_DOUBLE
glEdgeFlagPointer	1	no type argument (type of data must be GLboolean)
glFogCoordPointer	1	GL_FLOAT, GL_DOUBLE

Stride

- With a stride of zero
  - Each type of vertex array must be tightly packed and homogeneous

- When dealing with interleaved arrays:
  - Using a stride of other than zero
  - Example (both color and position):

```
static GLfloat intertwined[]={1.0, 0.2, 1.0, 100.0, 100.0, 0.0,
                              1.0, 0.2, 0.2, 0.0, 200.0, 0.0,
                              1.0, 1.0, 0.2, 100.0, 300.0, 0.0,
                              0.2, 1.0, 0.2, 200.0, 300.0, 0.0,
                              0.2, 1.0, 1.0, 300.0, 200.0, 0.0,
                              0.2, 0.2, 1.0, 200.0, 100.0, 0.0};

glColorPointer (3, GL_FLOAT, 6 * sizeof(GLfloat), &intertwined[0]);
glVertexPointer(3, GL_FLOAT,6*sizeof(GLfloat), &intertwined[3]);
```

Example

```
//    v6----- v5
//   /|      /|
//  v1------v0|
//  | |     | |
//  | |v7---|-|v4
//  |/      |/
//  v2------v3
```

```
// vertex coords array
GLfloat vertices[] = {1,1,1,  -1,1,1,  -1,-1,1,  1,-1,1,        // v0-v1-v2-v3
                      1,1,1,  1,-1,1,  1,-1,-1,  1,1,-1,        // v0-v3-v4-v5
                      1,1,1,  1,1,-1,  -1,1,-1,  -1,1,1,        // v0-v5-v6-v1
                      -1,1,1,  -1,1,-1,  -1,-1,-1,  -1,-1,1,    // v1-v6-v7-v2
                      -1,-1,-1,  1,-1,-1,  1,-1,1,  -1,-1,1,    // v7-v4-v3-v2
                      1,-1,-1,  -1,-1,-1,  -1,1,-1,  1,1,-1};   // v4-v7-v6-v5

// normal array
GLfloat normals[] = {0,0,1,  0,0,1,  0,0,1,  0,0,1,           // v0-v1-v2-v3
                     1,0,0,  1,0,0,  1,0,0, 1,0,0,            // v0-v3-v4-v5
                     0,1,0,  0,1,0,  0,1,0, 0,1,0,            // v0-v5-v6-v1
                     -1,0,0,  -1,0,0, -1,0,0,  -1,0,0,        // v1-v6-v7-v2
                     0,-1,0,  0,-1,0,  0,-1,0,  0,-1,0,       // v7-v4-v3-v2
                     0,0,-1,  0,0,-1,  0,0,-1,  0,0,-1};      // v4-v7-v6-v5

// color array
GLfloat colors[] = {1,1,1,  1,1,0,  1,0,0,  1,0,1,           // v0-v1-v2-v3
                    1,1,1,  1,0,1,  0,0,1,  0,1,1,           // v0-v3-v4-v5
                    1,1,1,  0,1,1,  0,1,0,  1,1,0,           // v0-v5-v6-v1
                    1,1,0,  0,1,0,  0,0,0,  1,0,0,           // v1-v6-v7-v2
                    0,0,0,  0,0,1,  1,0,1,  1,0,0,           // v7-v4-v3-v2
                    0,0,1,  0,0,0,  0,1,0,  0,1,1};          // v4-v7-v6-v5
```

```
Void drawFunction()
{
   // enble and specify pointers to vertex arrays
   glEnableClientState(GL_NORMAL_ARRAY);
   glEnableClientState(GL_COLOR_ARRAY);
   glEnableClientState(GL_VERTEX_ARRAY);

   glNormalPointer(GL_FLOAT, 0, normals);
   glColorPointer(3, GL_FLOAT, 0, colors);
   glVertexPointer(3, GL_FLOAT, 0, vertices);

   glDrawArrays(GL_QUADS, 0, 24);

   glDisableClientState(GL_VERTEX_ARRAY);
   glDisableClientState(GL_COLOR_ARRAY);
   glDisableClientState(GL_NORMAL_ARRAY);
}
```

rendering

Step 2 - Rendering

- **glDrawArrays()**
  - Reads vertex data from the enabled arrays by marching straight through the array without skipping or hopping.
    - Have to repeat the shared vertices once per face.
    - 3 arguments:
      - the primitive type.
      - the starting offset of the array.
      - the number of vertices
  - Example:

```
…
glEnableClientState(GL_VERTEX_ARRAY);
glVertexPointer(3, GL_FLOAT, 0, vertices);
glDrawArrays(GL_QUADS, 0, 24);
…
```

Step 2 - Rendering I (Single Array Element)

- *void **glArrayElement**(GLint ith)*
  - *Obtains the data of one (the ith) vertex for all currently enabled arrays.*
  - Is usually called between **glBegin()** and **glEnd()**
  - is good for randomly "hopping around" your data arrays
- Example -

```
...
glColorPointer(3,GL_FLOAT, 0, colors);
glVertexPointer(2,GL_INT, 0, vertices);
glBegin(GL_TRIANGLES);
glArrayElement (2);
glArrayElement (3);
glArrayElement (5);
glEnd();
```

```
///Same as:
glBegin(GL_TRIANGLES);
glColor3fv(colors+(2*3*sizeof(GLfloat));
glVertex3fv(vertices+(2*2*sizeof(GLint));
glColor3fv(colors+(3*3*sizeof(GLfloat));
glVertex3fv(vertices+(3*2*sizeof(GLint));
glColor3fv(colors+(5*3*sizeof(GLfloat));
glVertex3fv(vertices+(5*2*sizeof(GLint));
glEnd();
```

Step 2 - Rendering

- **glDrawArrays()**

```
…
glEnableClientState(GL_VERTEX_A
   RRAY);
glVertexPointer(3, GL_FLOAT, 0,
   vertices);
glDrawArrays(GL_QUADS, 0, 24);
…
```

- **glArrayElement()**

```
int i;
glBegin (mode);
for (i = 0; i < count; i++)
        glArrayElement (vertices + i);
glEnd();
```

Step 2 - Rendering I (Sequence of Array Element)

- **glDrawElements()**
  - draws a sequence of primitives by hopping around vertex arrays with the associated array indices.
  - reduces
    - the number of function calls
    - the number of vertices to transfer.
  - OpenGL may cache the recently processed vertices and reuse them without resending the same vertices into vertex transform pipeline multiple times.
  - 4 parameters.
    - type of primitive,
    - number of indices of index array,
    - data type of index array
    - the address of index array.

Example

```
static GLubyte frontIndices = {4, 5, 6, 7};
static GLubyte rightIndices = {1, 2, 6, 5};
static GLubyte bottomIndices = {0, 1, 5,4};
static GLubyte backIndices = {0, 3, 2, 1};
static GLubyte leftIndices = {0, 4, 7, 3};
static GLubyte topIndices = {2, 3, 7, 6};

glDrawElements(GL_QUADS, 4, GL_UNSIGNED_BYTE,
   frontIndices);
glDrawElements(GL_QUADS, 4, GL_UNSIGNED_BYTE,
   rightIndices);
glDrawElements(GL_QUADS, 4, GL_UNSIGNED_BYTE,
   bottomIndices);
glDrawElements(GL_QUADS, 4, GL_UNSIGNED_BYTE,
   backIndices);
glDrawElements(GL_QUADS, 4, GL_UNSIGNED_BYTE,
   leftIndices);
glDrawElements(GL_QUADS, 4, GL_UNSIGNED_BYTE,
   topIndices);
```

Or :

```
static GLubyte allIndices = {4, 5, 6, 7,
                             1, 2, 6, 5,
                             0, 1, 5, 4,
                             0, 3, 2, 1,
                             0, 4, 7, 3,
                             2, 3, 7, 6};
glDrawElements(GL_QUADS,24, GL_UNSIGNED_BYTE,
   allIndices);
```

3rd Task

Vertex Array

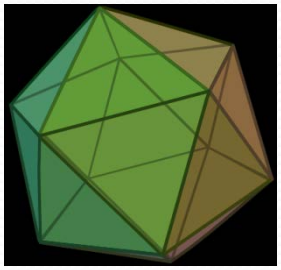
1. Create
   - vertex array that describe the tetrahedron you previously made

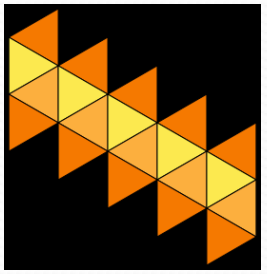
     GLfloat vertices[] = {1,1,1, -1,1,1, ...};
   - color array

     GLfloat colors[] = {1,1,1, 1,1,0, 1,0,0, ...};
   - index array of vertex array for glDrawElements()
1. Draw your figure with glDrawArrays()
2. Translate your new figure to the left top corner
3. Repeat steps 1 to 4 for drawing a third figure using glDrawElements
4. Translate last figure to the right top corner

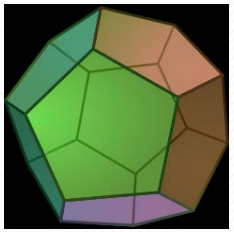
Bonus

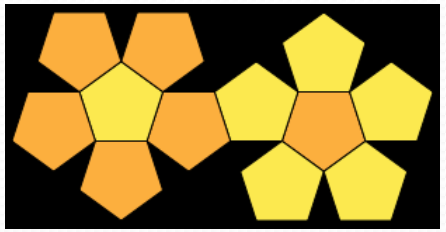
1. Draw a **Icosahedron**
   - Vertices :(0, ±1, ±ϕ) (±1, ±ϕ, 0) (±ϕ, 0, ±1) where ϕ = (1+√5)/2





2. Draw **Dodecahedron**
   - Vertices :(±1, ±1, ±1) (0, ±1/ϕ, ±ϕ) (±1/ϕ, ±ϕ, 0) (±ϕ, 0, ±1/ϕ) where ϕ = (1+√5)/2





3. Make an a creative 3D scene with Vertex Array

- Write you name and student number in comments at the top of your code
- Duplicate your .cpp
- Name it lab1_First_Last.cpp
- Submit your code as follows:

    submit 4431 lab1 filename(s)