Today's Agenda

> Check your github repos

> C programming basics

> Vertex Arrays

> GLU primitives

> Basic Camera - keyboard and mouse movements

C Programming

A variable is the address to a space in memory.

Depending on the the type of variable, the space has a particular size.

the size of a particular type varies from architecture to architecture, but in general:

ints are 16 bits
    unsigned = -32768 -> +32768
    signed = 0 -> 65535
    (by default signed)

longs are 32 bit integers

floats are 32 bits

doubles are 64 bits

chars are 8 bits

C Programming

In many cases, you simply need the item inside the memory that the variable refers to.

```
int x = 5;
int y = 11;
int z = x + y;
printf("z = %d \n", z);
```

In other cases it is useful to know the address of that memory. For this you can use the & operator.

```
int *a = &z;
```

The variable "a" is a **pointer** to the address where the integer value of z lives.

A pointer is declared with an *

The indirection operator is also an *

because our variable a is an "int pointer" we can use the indirection operator to see what is inside the address that a points to...

```
printf("a points to %d\n", *a);
```

C Programming

In C, all variables passed to a function are copied into that function.

```
void foo(int a) {
    a += 1;
}

int main(int argc, char** argv) {
    int a = 5;
    foo(a);
    printf("a now = %d\n", a); //a = 5;
}
```

However, you can pass a pointer into the function...

```
void foo(int *a) {
    *a += 1;
}

int main(int argc, char** argv) {
    int a = 5;
    foo(&a);
    printf("a now = %d\n", a); //a = 5;
}
```

C Programming

You can also create your own simple data types using structs

```
struct info {
    int i;
    float f;
    char *s;
};

struct info info1;
info1.i = 10;
info1.f = 9.99;
info1.s = (char *) malloc (sizeof(char) * 4);
info1.*s = "abc";
```

GLU geometry

```
//GLU

GLUquadricObj qObj = gluNewQuadric(); //load up new quadric
gluQuadricDrawStyle(qObj, GLU_FILL); //fill in the facets on the surface
gluQuadricNormals(qObj, GLU_SMOOTH); //make nice normals across the facets
//if lighting is enabled then the normals are already present
gluQuadricTexture(qObj, true); //automatically create texture coords for the object
//if you a bind a texture, then the texture coords are already present

gluSphere(qObj, 2f, 64, 64); //make a sphere with radius 2 and high resolution

//check out the glu library for other primitives
```

Vertex Arrays

Vertex Array are generally much faster than immediate mode. Newer version of OpenGL do *not* support immediate mode at all.

The idea is that you send a chunk of buffered data all at once, rather than a single vertex (i.e. immediately) one at a time. Since the throughput from the CPU to the GPU can be the bottleneck if you have a lot of polygons to render, this can increase the framerate of your program.

init:

1. Set up your arrays (or a space in memory to hold your arrays).

in render loop:

2. Initialize or update the arrays with the vertex information, if necessary
3. Enable vertex arrays for colors, normals, texture coords, and vertices, as needed
4. Define a pointer and offset into each of the arrays, along with the appropriate "stride"
5. Pass the arrays to the GPU
6. Disable the vertex arrays

I'll probably alternate between immediate mode and vertex arrays throughout the class. Newer versions of OpenGL also support vertex buffer objects (VBOs) and vertex array objects (VAOs).

Vertex Arrays - simple 1 triangle example

```
//global data arrays
float c[] = { 1.0, 0.0, 0.0, 1.0,     0.0, 1.0, 0.0, 1.0,   0.0, 0.0, 1.0, 1.0 };
float t[] = { 0.0, 0.0,               1.0, 0.0,             0.5, 1.0 };
float n[] = { 0.0, 0.0, 1.0,          0.0, 0.0, 1.0,        0.0, 0.0, 1.0 };
float v[] = { -1.0, -1.0, 0.0,        1.0, 1.0, 0.0,        0.0, 1.0, 0.0 };

//enable each type of array (the only one that is *required* is GL_VERTEX_ARRAY)
glEnableClientState(GL_COLOR_ARRAY);
glEnableClientState(GL_TEXTURE_COORD_ARRAY);
glEnableClientState(GL_NORMAL_ARRAY);
glEnableClientState(GL_VERTEX_ARRAY);

//point into the each of the enabled arrays
glColorPointer(4, GL_FLOAT, 0, c); //4 because using RGBA
glTexCoordPointer(2, GL_FLOAT, 0, t); //2 because using 2D texture
glNormalPointer(3, GL_FLOAT, 0, n); //3 because using 3D normal vector
glVertexPointer(3, GL_FLOAT, 0, v); //3 because using 3D points

//send the data in one big chunk
glDrawArrays(GL_TRIANGLES, 0, 1); //type of primitive, offset, number of each primitive

//disable each type of array
glDisableClientState(GL_COLOR_ARRAY);
glDisableClientState(GL_TEXTURE_COORD_ARRAY);
glDisableClientState(GL_NORMAL_ARRAY);
glDisableClientState(GL_VERTEX_ARRAY);
```

Assignments

Make the landscape a height map... show how to load this in

Write code that correctly textures your landscape (either as a quad strip or triangle strip) using one or more images. Image you are writing a game where you are a tank, and you have a "first-person" view of the landscape from atop your tank. Write a simple camera that lets you move forward and backward, and to turn left or right by a small amount when you press the left and right keys.