

# Lab 3 Implementation Tips

Comp 360 - Tutorial 2

10/18/2009

Poweï Feng

# Lab 3

- The bad news: No framework other than the very basic one. Boo~!
- The good news: You'll gain a lot of experience in programming. Yay~!
- How should we tackle this project?

# My recommendation

- Use some simple OO
  - Have one abstract "GeometricObject" class
    - GO has a virtual function call "intersect"
    - Maybe a function call "getNormal"
    - Represent each geometric objects as separate classes, who are also the subclasses of "GeometricObject"
  - Create a class for light

# My recommendation

- Create a raytracer class that contains a list of GO's and lights
  - Keep a pixel buffer in raytracer
  - Write your trace function in raytracer
  - Update one pixel at a time
  - Pass the buffer to `glDrawPixel` in the `renderFunc()` function in Callback

# Some final tips

- Create math utility classes such as Vector and Matrix
  - You could have Vector3 and Vector4 to represent vectors of 3 and 4 components
  - Color could be a subclass or alias of Vector3 (or 4)
  - Define simple vector-vector vector-matrix operations
- Use a small screen size/coarser resolution for testing
  - Resolution can be changed with glPixelZoom
  - Screen size is specified in OpenGLFrame.h
- Since our computation is purely software and shouldn't be limited by graphics card's refresh rate. Don't just compute a single pixel in your renderFunc(). Do it for multiple pixels and then render. (This is a little contradictory to the project description.)

# One Last Cool C++ Trick

Given a point  $p$  and a vector  $v$ . We write the point  $q=p+v$ . Wouldn't it be great if we can code it as the following?

```
Vector v(1,2);  
Point p(3,4);  
Point q = p + v;
```

# One Last Cool C++ Trick

Given a point  $p$  and a vector  $v$ . We write the point  $q=p+v$ . Wouldn't it be great if we can code it as the following?

```
Vector v(1,2);  
Point p(3,4);  
Point q = p + v;
```

In fact, you can. C++ provides a nice functionality call "operator overloading." It means that you can define specific functionality for the "+" operator (or one of many others) of any class you wish.

# Operator Overloading

```
class Point{  
    ...  
    Point operator+(const Vector& v) const{  
        return Point(_val[0]+v.getX(),_val[1]+v.getY());  
    }  
    ...  
};
```

- This is very useful for reasoning about math formulas. You can overload "\*" for dot product etc.
- Unfortunately, operator overloading requires very careful implementation. For examples, please consult
  - <http://www.cs.caltech.edu/courses/cs11/material/cpp/donnie/cpp-ops.html>
  - <http://www.cs.caltech.edu/courses/cs11/material/cpp/mike/>