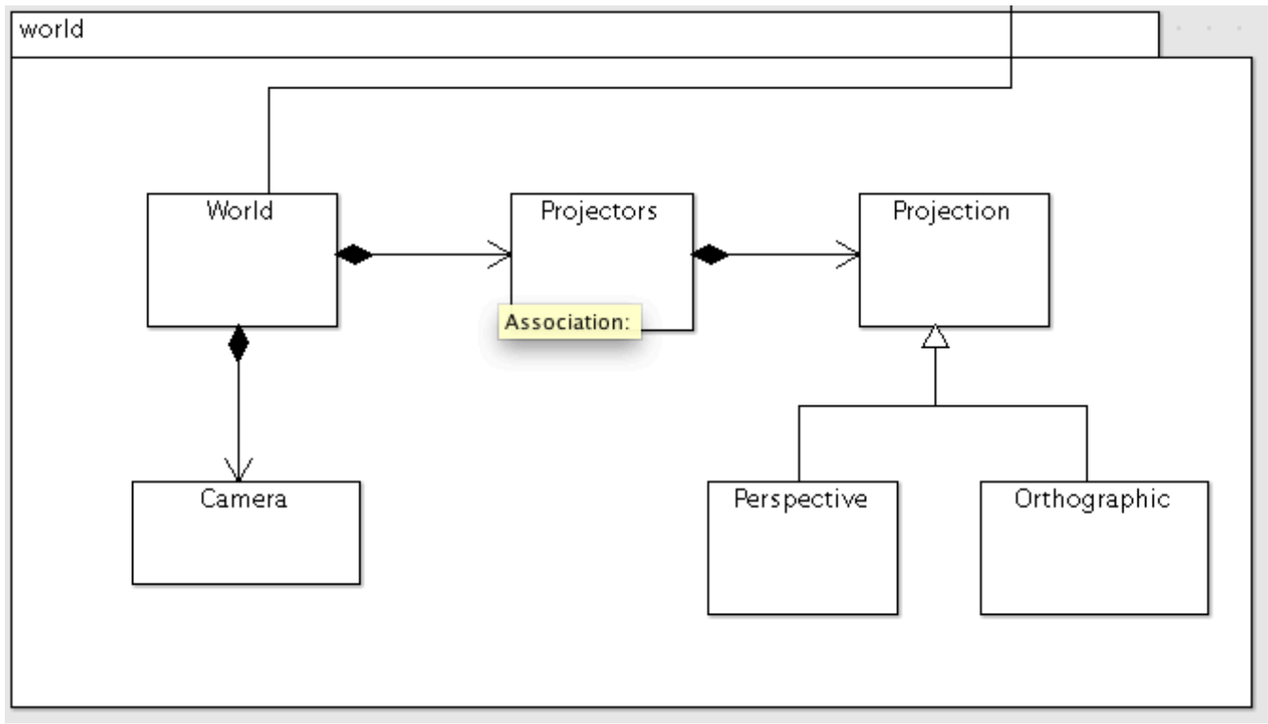
Camera Class

OpenGL

Camera Class


```
struct Camera
{
  Camera();
  void keyStroke (unsigned char key);
  void render();

  float xrot, yrot;
  Vector3 position;
  float controlPrecision;
};
```



Camera Implementation (first person)

```
Camera::Camera()
: position (0,0,-10)
{
}

void Camera::render()
{
  Vector3::UnitX.rotate(xrot);
  Vector3::UnitY.rotate(yrot);
  position.translate();
}
```

- 22. OpenGL Camera March 25, 2010

If you look at making any game where the scene is larger than can be displayed in the window at once, then you are going to need some type of camera system. This is the first of several tutorials on building a first person camera system.

- 23. OpenGL Camera Part 2 March 25, 2010

Here I will be extending upon the previous OpenGL Camera tutorial, and adding a strafe feature (moving side to side). A game without strafing, is going to be terrible, especially when it comes to multiplayer when strafing against enemy fire is essential.

```
void Camera::keyStroke (unsigned char key)
{
  if (key == 'q')
  {
    if (xrot > 360)
      xrot -= 360;
  }
  else if (key == 'z')
  {
    if (xrot < -360)
      xrot += 360;
  }
  else if (key == 'd')
  {
    if (yrot > 360)
      yrot -= 360;
  }
  else if (key == 'a')
  {
    if (yrot < -360)
      yrot += 360;
  }
  else if (key == 'w' || key == 's')
  {
    float xrotrad, yrotrad;
    yrotrad = (yrot / 180 * GL_PI);
    xrotrad = (xrot / 180 * GL_PI);
    if (key == 'w')
    {
      position.X += float(sin(yrotrad));
      position.Y -= float(sin(xrotrad));
      position.Z -= float(cos(yrotrad))  ;
    }
    else if (key == 's')
    {
      position.X -= float(sin(yrotrad));
      position.Y += float(sin(xrotrad));
      position.Z += float(cos(yrotrad)) ;
    }
  }
}
```

Camera With Mouse + Arrow Keys

```
struct Camera
{
  Camera();

  void specialKeyboard (int key, int x, int y);
  void mouseMovement(int x, int y);
  void render();

  Vector3 position;
  float xrot, yrot, cRadius, lastx, lasty;
};
```

Glut Support - Special Keys (arrow, fn..)

7.9 glutSpecialFunc

`glutSpecialFunc` sets the special keyboard callback for the *current window*.

Usage

```
void glutSpecialFunc(void (*func)(int key, int x, int y));
```

`func`

The new entry callback function.

Description

`glutSpecialFunc` sets the special keyboard callback for the *current window*. The special keyboard callback is triggered when keyboard function or directional keys are pressed. The `key` callback parameter is a `GLUT_KEY_*` constant for the special key pressed. The `x` and `y` callback parameters indicate the mouse in window relative coordinates when the key was pressed. When a new window is created, no special callback is initially registered and special key strokes in the window are ignored. Passing `NULL` to `glutSpecialFunc` disables the generation of special callbacks.

Glut Support - mouse

7.6 glutMotionFunc, glutPassiveMotionFunc

`glutMotionFunc` and `glutPassiveMotionFunc` set the motion and passive motion callbacks respectively for the *current window*.

Usage

```
void glutMotionFunc(void (*func)(int x, int y));
void glutPassiveMotionFunc(void (*func)(int x, int y));
```

`func`

The new motion or passive motion callback function.

Description

`glutMotionFunc` and `glutPassiveMotionFunc` set the motion and passive motion callback respectively for the *current window*. The motion callback for a window is called when the mouse moves within the window while one or more mouse buttons are pressed. The passive motion callback for a window is called when the mouse moves within the window while *no* mouse buttons are pressed.

The `x` and `y` callback parameters indicate the mouse location in window relative coordinates.

Passing `NULL` to `glutMotionFunc` or `glutPassiveMotionFunc` disables the generation of the mouse or passive motion callback respectively.

Extend World

- Augment glutadapter

```
void mouseMovement(int x, int y)
{
  theWorld.mouseMovement(x,y);
}

void keyboardSpecial(int key, int x, int y)
{
  theWorld.specialKeyPress(key, x, y);
}
```

```
struct World
{
  void initialize(std::string name, int width, int height);

  void keyPress(unsigned char ch);

  void specialKeyPress(int key, int x, int y);
  void mouseMovement(int x, int y);

  void start();
  void render();
  void tickAndRender();

  static World& GetInstance();
  static World  *s_World;

  Scene          *scene;
  Projectors      projectors;
  Camera          camera;
};
```

World Implementation

```
void World::render()
{
  glClearColor(0.0, 0.0, 0.0, 1.0);
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  if (projectors.isPerspective())
  {
    glLoadIdentity();
    camera.render();
  }
  scene->render();
  glutSwapBuffers();
}
```

```
void World::keyPress(unsigned char ch)
{
  if (ch >= '1' && ch <= '4')
  {
    projectors.keyPress(ch);
  }
  glutPostRedisplay();
}

void World::specialKeyPress(int key, int x, int y)
{
  if (projectors.isPerspective())
  {
    camera.specialKeyboard(key, x, y);
  }
  glutPostRedisplay();
}

void World::mouseMovement(int x, int y)
{
  camera.mouseMovement(x,y);
}
```

```
void World::initialize(string name, int width, int height)
{
  //...
  glutSpecialFunc(keyboardSpecial);
  glutPassiveMotionFunc(::mouseMovement);
}
```

Camera - 3rd Person

- It should be a set distance from a certain subject. This set distance could of course change if desired.

- It should also be a certain amount along the circumference of a circle that has that subject as its center.

```
struct Camera
{
  Camera();

  void specialKeyboard (int key, int x, int y);
  void mouseMovement(int x, int y);
  void render();

  Vector3 position;
  float xrot, yrot, cRadius, lastx, lasty;
};
```

3rd Person Camera Implementation

```
Camera::Camera()
: position(0,0,0), xrot(0), yrot(0), cRadius(5)
{
}

void Camera::render()
{
  glTranslatef(0.0f, 0.0f, -cRadius);
  Vector3::UnitX.rotate(xrot);
  Vector3::UnitY.rotate(yrot);
  glTranslated(-position.X, 0.0f, -position.Z);
}


void Camera::mouseMovement(int x, int y)
{
  int diffx = x - lastx;
  int diffy = y - lasty;
  lastx = x;
  lasty = y;
  xrot += (float) diffy;
  yrot += (float) diffx;
}
```

```
void Camera::specialKeyboard(int key, int x, int y)
{
  float xrotrad, yrotrad;
  switch (key)
  {
    case 101: yrotrad = (yrot / 180 * GL_PI);
              xrotrad = (xrot / 180 * GL_PI);
              position.X += float(sin(yrotrad));
              position.Z -= float(cos(yrotrad));
              position.Y -= float(sin(xrotrad));
              break;

    case 103: yrotrad = (yrot / 180 * GL_PI);
              xrotrad = (xrot / 180 * GL_PI);
              position.X -= float(sin(yrotrad));
              position.Z += float(cos(yrotrad));
              position.Y += float(sin(xrotrad));
              break;

   case 102: yrotrad = (yrot / 180 * GL_PI);
              position.X += float(cos(yrotrad)) * 0.2;
              position.Z += float(sin(yrotrad)) * 0.2;
              break;

   case 100: yrotrad = (yrot / 180 * GL_PI);
              position.X -= float(cos(yrotrad)) * 0.2;
              position.Z -= float(sin(yrotrad)) * 0.2;
              break;
}
```

- 24. OpenGL Camera Part 3 March 25, 2010

The first person camera is done, lets take a look at the third person camera, which is essential any type of Role Playing Game. The best part is, this tutorial uses most of the same code as the previous camera tutorial. Just some minor changes to entirely change the feel of your game.

Exercise 1

- You will have seen that the camera in this lab adopts a 'third person' type perspective - and was influenced by the geometry calculations in this tutorial here:

  - http://www.swiftless.com/tutorials/opengl/camera3.html

- Our earlier camera was influenced these one here:

  - http://www.swiftless.com/tutorials/opengl/camera.html

- You have to code for both these camera implementations. As we did with Projections - incorporate both classes into a simple class hierarchy - and adopt some key to switch cameras -'c' say. How much of the camera implementations can be retained in a camera base class - and how much in the derived classes?

Exercise 1 Model