3D Object Oriented OpenGL® Graphics Engine

Jonathan Snelling
CS-470 Project Write-up

2006-05-02

Table of Contents

Abstract

The 3D Object Oriented OpenGL® Graphics Engine accelerates and simplifies the development and deployment of three dimensional graphics applications. The projects provides an object oriented approach to graphics development taking inspiration from 2D graphics frameworks like GTK or Swing.

1. Introduction

During the course of this semester I designed and demonstrated a graphics engine which I created to aid in the rapid design and deployment of graphical applications requiring hardware supported 3d real-time rendering. Games are a good example of such an application but there are many other applications such as mathematical or scientific visualization and demonstration.

2. Project Overview

Though this project was designed and implemented to satisfy the requirements CS470 at the University of Alaska, I have already found interest from a coworker who would be interested in seeing a video game designed and driven by my API. I also have further plans for usage of my project to deploy demonstration programs to visually represent advanced mathematical and scientific theories.

The goal of this project was to develop an OpenGL driven graphics engine to simplify and accelerate the development and deployment of 3d graphics visualization and entertainment. The key improvement provided by this project in be the imposing of object-oriented representation in the graphics engine.

Though the system will be object oriented, it will be implemented in GNU standard C++, which though object oriented, runs in a machine native, high-speed form.

2.1 Data Files

For convenience, the engine supports the Wavefront .obj file format which is a standard mesh and object storage data format. I implemented this support myself as part of my project. My code currently supports a fair subset of the .obj file format and support will increase in future versions.

Compatibility with common image file formats is provided by use of the DevIL (OpenIL) graphics library. Support for .bmp, .cut, .dcx, .dds, .ico, .gif, .jpg, .lbm, .lif, .mdl, .pcd, .pcx, .pic, .png, .pnm, .psd, .psp, .raw, .sgi, .tga, .tif, .wal, .act .pal, .hdr and Doom graphics are supported. Non power of two textures may be used since the engine will automatically scale the image returned by DevIL to the nearest smaller power of 2.

As I'd hoped, integration with the popular Open-Source 3d design and rendering package, Blender is a simple reality. Blender has complete support for a similar subset of the .obj file format that my engine implements. This similarity is no accident on my part and allows for objects exported from blender to be used in my engine.

3. Project Requirements

Since this project was produced without a specific initial client, the requirements were up to me. I consider the project to be a complete success since I have created an object oriented engine driven by OpenGL that includes support for selected peripheral as well as 'picking.'

The engine also have support for multiple resolutions (through resizing, finalization of initial resolution support options are coming) as well as full screen mode. The engine support the previously discussed networking is a seamless transparent support that makes it simple for programmers to offer networking in their projects based on my API. Sadly the network support is at a limited state, but it will become more refined in further revisions.

Additionally I have included a series of demos and tutorials designed to showcase the capabilities of the engine. These demo's are currently a little clunky to access. I will create and include a demonstration object that contains all of the demo's and gives I smooth way to access them in future versions.

3.1 Functional Specifications

1. To begin, the headers for the engine are included in the project.

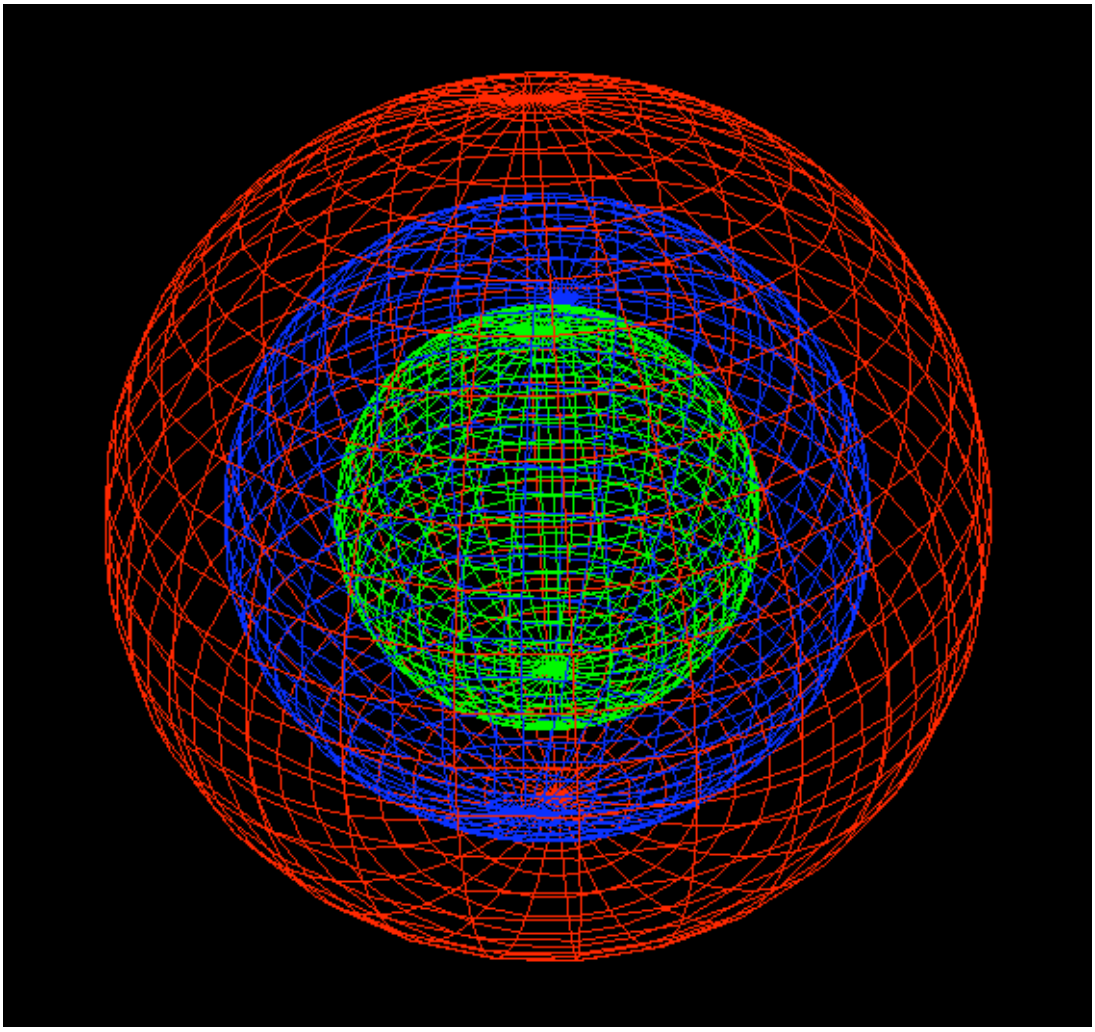
2. The coder extends one of the existing classes to create the object they need for their project. Even the simplest base class (jObject) contains the ability to support key press detection and mouse click detection.

2. The application is setup by creating a JWorld and adding objects.

Example:

```
#include "module.h"
#include "extended.h"

int main(int argc, const char **argv) {

      jWorld world = jWorld();

    jCartesian mapping = jCartesian();
    float den = 20;
    for (float j = -den; j <= den; j ++) {
        for (float i = -den; i <= den; i ++) {
            jSpinner *spinner = new jSpinner(4/den);
            mapping.add(spinner,i*4/den,j*4/den,0);
            spinner->setColor(((float)i+den)/(2*den),((float)j+den)/
(2*den),1);
                  world.registerName(spinner);
            world.registerKey(spinner,'a'+((int)((j+den)*(i+den)+(i
+den)))%26);
            }
      }

      world.add(&mapping);
      world.goGL(argc,argv);
    return 0;
}
```

3. Finially the jWorld is sent a goGL command and the program runs.



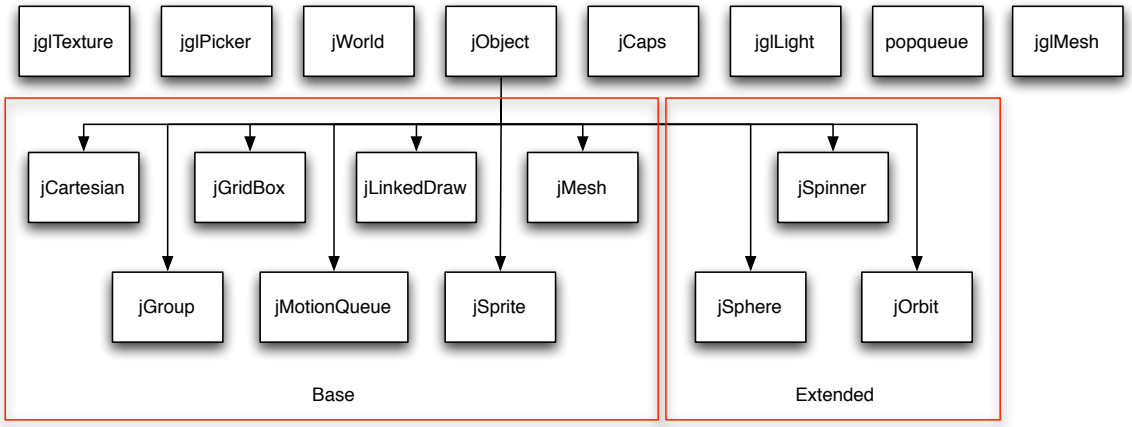
3.2 System Specifications

The engine is designed to run on POSIX systems, especially the world's most advanced operating system: Mac OS X. The engine should be compilable on POSIX system, but benefits from the advanced programming frameworks for collection handling provided by Apple's CoreFoundation when present.

There has been request that the engine be made to be compatible with Windows... Should I find a programmer who's sadistic enough to assist with that request, I probably will not resist too much.

4. System Design

The modular system would be difficult to describe, so I'll let the following diagram speak for itself.

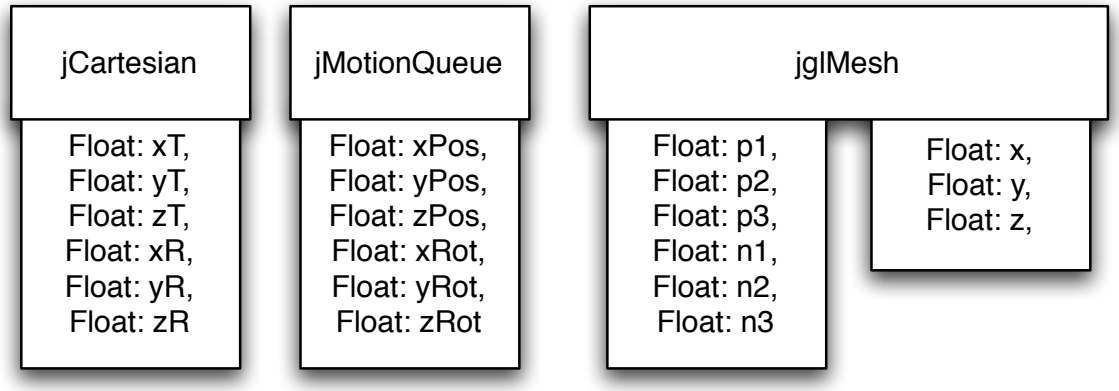


4.1 User Interface Design

For now any user interface present must be implemented by the end programmer. In the future I plan to release widgets for my engine containing two and three dimensional buttons, text fields, check boxes and other interface elements.

4.2 Data Structures

Notable data structures used in the project can be found below.



The structures found in jCartesian and jMotionQueue are very similar and just represent the position and orientation of an object in three dimensional cartesian space. there are two data structures used by the jglMesh helper object. The first of the two data structures contains references to three points and three normals that make up a face. The second data structure is used to keep two lists: one for the points in the object and one for the normals in the object. The array of faces is then stepped through using the other two arrays for lookup and a glCallList is used to compile the object for quicker draw.

4.3 Engine Architecture

The engine is based off of the idea of doing automatic setup and then using a series of object oriented entries to draw a scene. The method is further discussed below.

The included objects are currently broken up into two sections. There are base objects which implement core functionality such as Mesh basis or purely delegated draw, update, ect. The extended base currently include more specialized objects which inherit from a base object. In the near future I intend to create a set of user interface widgets. I am currently unsure of whether those widgets will find there way into the extended set or whether they will reside in their own section.

4.4 Algorithms

There are a series of algorithms that I wrote for the engine to function. The most basic of the algorithms is the structure used to cascade calls for drawing and updating to added children objects.

The jWorld object contains one single jObject. When the jWorld is asked by GLUT to draw the scene or to do an update, it calls drawObject or updateObject on it's jObject. Each jObject or jObject inheritor contains a mutable array which contains pointers to each of the children that have been added to the object. When the object begins a draw, it applies any transformations that may be applicable to it's object and then calls drawObject on each of it's children. Finally the object draws itself and undoes the transformations.

Due to the potentially huge number of objects that may exist in a given object, and the fact that OpenGL implementations only guarantee 32 items can be stored on the stack, I decided to avoid use of glPushMatrix() and glPopMatrix() and instead all my objects invert the sequence of transformations from the beginning of the draw function at the end. This decision means that the object tree's can greatly exceed the stack limit of the OpenGL implementation.

Another algorithm that I use is a method for referencing external objects such as meshes or textures by name. When a coder needs access to a texture or mesh, they just need to request it by name from the jWorld class. The jWorld hands off the request internally which leaves me open to change internal responsibility for jWorld services at my whim without any perceived change in API...

When the handler receives a request for an object, it loads the object into the system and returns a pointer to the resource for the coder to use. The neat part of the algorithm is that it then adds an entry to an internal array recording the name of the loaded object and the location it is stored in. The functionality allows the coder to be sure that they don't accidentally load multiple copies of a resource into memory. After the first time a resource is requested, the helper object merely finds it's entry and returns the address.

5.0 Software Development Process

I can only describe the software development process as FUN. There was a certain activation energy of sorts to get to a working framework and after that I developed the framework through creation of demos. Since there was no specific client and no specific requirements, the best way for me to determine what features I needed to include in my engine was to build a wide range of experimental projects using my engine. During the course of my projects, I would find features that I needed and add them.

Since my future plans for this project nearly certainly include community involvement I suspect that identical process will be utilized by the community for the continued development of the project.

5.1 Testing and Debugging

Another advantage to development using test projects is that it allowed for extensive testing of each feature that was added during the development processes. As a result the project has been well tested.

There was little to no debugging that I had to do with my project. The reason for this that that I wanted to devote all of my energy to the development of the best project possible, so I used C++ which I have years of experience in and so I didn't spend time 'trying to figure out how it was done in that language.'

5.2 Prototyping Challenges

The lack of a specific client for the project make it slightly tougher because I had to try to determine what a user was likely to need rather than just starting with a list of required features and implementing them.

My solution with the varying test projects allowed me to evolve a new engine prototype with each test project.

5.3 Work Breakdown

The majority of my time was spent playing around with my engine so that I could identify new features that I wanted to add. I really didn't start with an anticipated work schedule. I just built the first prototype and kept making revisions until the project was due.

6. Results

The engine turned out extremely well. I, and others, are very pleased with the development of the project and I am excited to continue the project. My project fulfilled all of the requirements that I started with and more.

Using my engine I was easily able to implement many different test applications, some of which will be included as capability demos in the project.

6.1 Future steps

I would like to involve the community in the continued development of the engine to speed the development of the process and to allow differing views to make the engine more universal. In the future I would like to include sound support, vertex and fragment shader support, better network support, and a user interface widget library.

7. Summary and Conclusions

In closing, I have been very pleased with my success in creating a graphics engine. I see a lot of potential for the current engine and even more potential for its continued development.

I have had a good time creating this engine and am eager to see how far I can take it. I have heard reports of my 401 project in use in Florida. My goal is to break that distance record with this one.

Appendix A: Preliminary API

jWorld:

void *add(jObject *adde);
adds a new object to the world

int addClickable(jObject *adde);
adds a new object to the world and registeres it to recieve mouse clicks

void *addWithKey(jObject *adde, char key);
adds a new object to the world and registers it to receive a notification for a key press.

int addClickableWithKey(jObject *adde, char key);
adds a new object to the world and registeres it to recieve mouse clicks as well as to receive a notification for a key press.

int static registerName(jObject *obj);
registers an object to recieve mouse clicks

bool static registerKey(jObject *obj,char key);
registers an object to receive a notification for a key press.

jObject:

virtual void draw();
called automatically when object is to be drawn

virtual void drawObject();
overloadable function defining how an object is drawn

virtual void update();
called automatically when object needs to be undated

virtual void updateObject();
overloadable function defining how an object is updated

virtual void clickedObject(int name);
called by jWorld when the object is clicked on

virtual void keypressNotify(char key);
called by jWorld when a key that the object has registed as a listener for has been pressed

virtual void *add(jObject *adde);
adds an object a a child of the current object.

virtual bool isVisible();
checks the current visibility of an object

void showMe();
reveals the current object

void hideMe();
hides the current object

void show();
reveals the object and ALL CHILDREN

void hide();
hides the object and ALL CHILDREN

virtual void setName(int name);
called automatically by jWorld to inform the object that it has been registed to receive mouse clicks, the name passed is the identifier assigned to the object by jWorld

void setClickDelegate(void (*clickFunction)(int name));
defines an external function to be called by the object when it is clicked on.

jMotionQueue:

void moveAbsolute(float xpos, float ypos, float zpos, float xrot, float yrot, float zrot, float duration);
moves the object to an abolute position and rotation in duration

void moveRelative(float xpos, float ypos, float zpos, float xrot, float yrot, float zrot, float duration);
moves the object by a relative translation and rotation in duration

void pause(float duration);
holds the object sill for duration before continuting.

jMesh:

jMesh(char *name);
creates a new mesh object based off of the name resource

jMesh(char *name, float Scale);
creates a new mesh object based off of the name resource and scales it by Scale.

Appendix B: Code Listing

Based on the early stage of development of a large scale project I have devoted more time to coding than to commenting. Later versions will be better commented.

popqueue.h

```
/*
 *  popqueue.h
 *  Module
 *
 *  Copyright 2005 Snelling All rights reserved.
 *
 *      Structure to implement a queue to push onto from start and end.
 */

#ifndef JON_POPQUEUE
#define JON_POPQUEUE


// structure to be linked to form a linked list
typedef struct {
        void                    *object;
        void                    *nextNode;
} popqueueNode;
```

```
//structure to house the nodes and keep information (length last node ect)
typedef struct {
	popqueueNode	*queue;
	int				length;
	popqueueNode	*last;
} popqueue;

popqueue	popqueueAlloc();

bool		enqueue(popqueue *theQueue, void *theObject);
void		*enqueueRetain(popqueue *theQueue, void *theObject, int size);
bool		push(popqueue *theQueue, void *theObject);
void		*pushRetain(popqueue *theQueue, void *theObject, int size);
void		*front(popqueue *theQueue);
void		*touch(popqueue *theQueue, int index);
void		*pop(popqueue *theQueue);
void		*dequeue(popqueue *theQueue);
#endif
```

popqueue.cpp

```
/*
 *  popqueue.c
 *  Module
 *
 *  Created by Jon Snelling, Oran Weaver, and Dave Schlerf on 9/18/05.
 *  Copyright 2005 Snelling et al. All rights reserved.
 *
 */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "popqueue.h"

popqueue popqueueAlloc() {
	popqueue p = {0,0,0};
	return p;
}

bool enqueue(popqueue *theQueue, void *theObject) {
	popqueueNode *temp;
	if (theQueue->length==0) {
		temp = (popqueueNode*)malloc(sizeof(popqueueNode));
		temp->object = theObject;
```

```
                temp->nextNode = NULL;
                theQueue->queue = temp;
                theQueue->length++;
                theQueue->last = temp;
                return true;
        }

        temp=theQueue->last;

        temp->nextNode = malloc(sizeof(popqueueNode));
        theQueue->last = (popqueueNode*)temp->nextNode;
        temp = (popqueueNode*)temp->nextNode;
        temp->object = theObject;
        temp->nextNode = NULL;
        theQueue->length++;
        return true;
}
void *enqueueRetain(popqueue *theQueue, void *theObject, int size) {
        void *temp;
        temp = malloc(size);
        memcpy(temp,theObject,size);
        enqueue(theQueue, temp);
        return temp;
}
bool push(popqueue *theQueue, void *theObject) {
        popqueueNode *temp;
        if (theQueue->length==0) {
                temp = (popqueueNode*)malloc(sizeof(popqueueNode));
                temp->object = theObject;
                temp->nextNode = NULL;
                theQueue->queue = temp;
                theQueue->length++;
                theQueue->last = temp;
                return true;
        }
        temp = (popqueueNode*)malloc(sizeof(popqueueNode));
        temp->object = theObject;
        temp->nextNode = theQueue->queue;
        theQueue->queue = temp;
        theQueue->length++;
        return true;
}
void *pushRetain(popqueue *theQueue, void *theObject, int size) {
        void *temp;
        temp = malloc(size);
        memcpy(temp,theObject,size);
```

```
	push(theQueue, temp);
	return temp;
}
void *front(popqueue *theQueue) {
	void *ret;

	if (theQueue->length==0) {
//		printf("Empty queue\n");
		return false;
	}
	ret = theQueue->queue->object;
	return ret;
}
void *touch(popqueue *theQueue, int index) {
	int i;
	popqueueNode *temp;

	temp = theQueue->queue;
	for (i=0; i<index; i++)
		temp = (popqueueNode*)temp->nextNode;
	return temp->object;
}
void *pop(popqueue *theQueue) {
//	popqueueNode *temp;
	void *ret;

	if (theQueue->length==0) {
//		printf("Empty queue\n");
		return false;
	}
	ret = theQueue->queue->object;

//	temp = theQueue->queue;
	theQueue->queue=(popqueueNode*)theQueue->queue->nextNode;
//	free(temp);
	theQueue->length--;

	return ret;
}
void *dequeue(popqueue *theQueue) {
	return pop(theQueue);
}
```

module.h

```
/*
```

```
 *  module.h
 *  Module
 *
 *  Created by Jonathan Snelling on 4/18/06.
 *  Copyright 2006 JumpMast3r. All rights reserved.
 *
 */

#include "jCartesian.h"
#include "jGridBox.h"
#include "jGroup.h"
#include "jLinkedDraw.h"
#include "jMesh.h"
#include "jMotionQueue.h"
#include "jObject.h"
#include "jSprite.h"
#include "jWorld.h"
```

extended.h

```
/*
 *  extended.h
 *  Module
 *
 *  Created by Jonathan Snelling on 4/18/06.
 *  Copyright 2006 JumpMast3r. All rights reserved.
 *
 */

#include "jSphere.h"
#include "jOrbit.h"
#include "jSpinner.h"
```

main.cpp

```
#include "module.h"
#include "extended.h"

int main(int argc, const char **argv) {

	jWorld world = jWorld();

	jCartesian mapping = jCartesian();
	float den = 20;
	for (float j = -den; j <= den; j ++) {
```

```
		for (float i = -den; i <= den; i ++) {
			jSpinner *spinner = new jSpinner(4/den);
			mapping.add(spinner,i*4/den,j*4/den,0);
			spinner->setColor(((float)i+den)/(2*den),((float)j+den)/(2*den),1);
			world.registerName(spinner);
			world.registerKey(spinner,'a'+((int)((j+den)*(i+den)+(i+den)))%26);
		}
	}

	world.add(&mapping);
	world.goGL(argc,argv);
	return 0;
}

//int main(int argc, const char **argv) {
//	jWorld world = jWorld();
//
//	jMotionQueue orbit2 = jMotionQueue(0,0,0);
//	jMotionQueue orbit = jMotionQueue(6.75,0,9);
//
//	jMesh mesh = jMesh("caveish.obj",1.5f);
//	orbit.add(&mesh);
//
//	orbit2.add(&orbit);
//	world.add(&orbit2);
//	srand(time(0));
//	for (int i =0 ; i< 50; i++)
//		orbit2.moveRelative(0,0,0,0,0,rand()%720-360,rand()%500);
//	orbit.moveRelative(0,0,0,0,3160,0,1800);
//
//	world.goGL(argc,argv);
//	return 0;
//}

//jMotionQueue orbit;
//void clicked(int name) {
//	orbit.moveRelative(0,0,0,0,360,0,30);
//}
//
//int main(int argc, const char **argv) {
//	jWorld world = jWorld();
//
//	orbit = jMotionQueue(0,0,0);
//
////	jSphere sphere = jSphere();
////	orbit.add(&sphere);
```

```
//
//        jSprite sprite = jSprite("Jonc.jpg",3.5f);
//        orbit.add(&sprite);
//
////      jMesh mesh = jMesh("JumpMast3r.obj",1.5f);
////      orbit.add(&mesh);
//
//        orbit.setClickDelegate(clicked);
//        world.addClickable(&orbit);
//
//        world.goGL(argc,argv);
//        return 0;
//}

//jMotionQueue bad1,bad2,bad3,bad4;
//int               name1,name2,name3,name4;
//void click(int name) {
//        if (name1==name) {
//                bad1.moveRelative(0,0,0,-90,0,0,4);
//                bad1.pause(rand()%100);
//                bad1.moveRelative(0,0,0,90,0,0,7);
//        } else if (name2==name) {
//                bad2.moveRelative(0,0,0,-90,0,0,4);
//                bad2.pause(rand()%100);
//                bad2.moveRelative(0,0,0,90,0,0,7);
//        } else if (name3==name) {
//                bad3.moveRelative(-1,0,0,0,0,90,4);
//                bad3.pause(rand()%100);
//                bad3.moveRelative(1,0,0,0,0,-90,7);
//        } else if (name4==name) {
//                bad4.moveRelative(0,0,0,90,0,0,4);
//                bad4.pause(rand()%200);
//                bad4.moveRelative(0,0,0,-90,0,0,7);
//
//        }
//}
//
//int main(int argc, const char **argv) {
//        jWorld world = jWorld();
//        srand(time(0));
//
//        jMesh mesh = jMesh("desert.obj",1.6f);
//        jMesh mesh2 = jMesh("badguy.obj");
//        jMesh mesh3 = jMesh("badguy.obj",0.5f);
//        jMesh mesh4 = jMesh("target.obj");
//
```

```
//        world.add(&mesh);
//
//        bad1 = jMotionQueue(3,0.5,0,-90,0,0);
//        bad1.pause(rand()%100);
//        bad1.moveRelative(0,0,0,90,0,0,7);
//
//        bad2 = jMotionQueue(-4,1,-3.5,-90,0,0);
//        bad2.pause(rand()%100);
//        bad2.moveRelative(0,0,0,90,0,0,7);
//
//        bad3 = jMotionQueue(-4,0,4,0,0,90);
//        bad3.pause(rand()%100);
//        bad3.moveRelative(1,0,0,0,0,-90,7);
//
//        bad4 = jMotionQueue(0,2.5,8,90,0,0);
//        bad4.pause(rand()%200);
//        bad4.moveRelative(0,0,0,-90,0,0,7);
//
//        bad1.setClickDelegate(click);
//        bad2.setClickDelegate(click);
//        bad3.setClickDelegate(click);
//        bad4.setClickDelegate(click);
//
//        bad1.add(&mesh2);
//        bad2.add(&mesh2);
//        bad3.add(&mesh3);
//        bad4.add(&mesh4);
//
//        name1 = world.addClickable(&bad1);
//        name2 = world.addClickable(&bad2);
//        name3 = world.addClickable(&bad3);
//        name4 = world.addClickable(&bad4);
//
//        world.goGL(argc,argv);
//        return 0;
//}

//float rot = 0;
//jMotionQueue spinna;
//int name1;
//int name2;
//
//void sph() {
//        glPushName(name2);
//        glTranslatef(-2,0,0);
//        glutWireSphere(1,20,20);
```

```
//        glColor3f(0,0,0);
//        glutSolidSphere(0.97 ,10,10);
//        glTranslatef(2,0,0);
//
//        glPopName();
//        glPushName(name1);
//        glColor3f(1,0,0);
//
//        jWorld::drawMesh("testOut.obj");
//        glTranslatef(0,3,0);
//        jWorld::drawMesh("test2.obj");
//        glTranslatef(0,-3,0);
//
//        glColor3f(0,1,1);
//        glTranslatef(2,0,0);
//        glutWireSphere(1,20,20);
//        glColor3f(0,0,0);
//        glutSolidSphere(0.97,10,10);
//        glTranslatef(-2,0,0);
//        glPopName();
//
//}
//void sphereUpdate() {
//        rot++;
//}
//void clickedObject(int name) {
//        printf("%d\n",name);
//        if (name==name1)
//                spinna.moveRelative(0,0,0,0,360,0,20);
//        else if (name==name2)
//                spinna.moveRelative(0,0,0,0,-360,0,20);
//}
//
//int main (int argc, const char **argv) {
//    jWorld world       = jWorld();
//        spinna = jMotionQueue();
////      spinna.moveAbsolute(0,0,0,0,0,90,10);
//        jLinkedDraw linked = jLinkedDraw(sph,sphereUpdate,0,clickedObject);
//
//
//        spinna.add(&linked);
//        name1 = world.registerName(&linked);
//        name2 = world.registerName(&linked);
//        world.add(&spinna);
//        world.goGL(argc,argv);
//
```

```
//    return 0;
//}

//int main(int argc, const char **argv) {
//
//        jWorld world = jWorld();
//
//        jGroup tiles = jGroup();
//        jCartesian mapping = jCartesian();
//        float den = 20;
//        for (float j = -den; j <= den; j ++) {
//                for (float i = -den; i <= den; i ++) {
//                        jSpinner *spinner = new jSpinner(4/den);
//                        mapping.add(spinner,i*4/den,j*4/den,0);
//                        spinner->setColor(((float)i+den)/(2*den),((float)j+den)/(2*den),1);
//                        world.registerName(spinner);
//                        world.registerKey(spinner,'a'+((int)((j+den)*(i+den)+(i+den)))%26);
//                }
//        }
//        tiles.add(&mapping);
//        world.registerKey(&tiles,';');
//        world.add(&tiles);
//        tiles.toggle();
//
//        jGroup motionQueue = jGroup();
//        jMotionQueue queue = jMotionQueue();
//        jSphere sphere = jSphere();
//        queue.add(&sphere);
//        motionQueue.add(&queue);
//        world.registerKey(&motionQueue,'\'');
//        world.add(&motionQueue);
//        queue.moveAbsolute(1,0,0,0,0,0,10);
//        queue.moveRelative(0,1,0,0,0,0,10);
//        queue.pause(10);
//        queue.moveAbsolute(1,0,0,0,0,0,20);
//        queue.moveRelative(0,1,0,0,0,0,20);
//        world.registerName(&queue);
//
//        world.goGL(argc,argv);
//        return 0;
//}

//int main(int argc, const char **argv) {
//        jWorld world = jWorld();
//
//        jOrbit orbit = jOrbit(2,2,1,5);
```

```
//        world.add(&orbit);
//
//        jSphere sphere1 = jSphere();
//        world.registerName(&sphere1);
//        world.add(&sphere1);
//
//        jSphere sphere2 = jSphere();
//        world.registerName(&sphere2);
//        orbit.add(&sphere2,-2,-2,-1,5);
//
//        world.goGL(argc,argv);
//        return 0;
//}

//float rot = 0;
//
//void sphereDraw(int r, int g, int b, float s,int n) {
//        glPushMatrix();
//        glColor3f(r,g,b);
//    glTranslatef(0.0f,0.0f,-6.0f);
//    glRotatef(n*rot,0.0f,0.5f,1.0f);
//    glutWireSphere(s,30,30);
//        glPopMatrix();
//}
//void sph() {
//        sphereDraw(1,0,0,2.0f,1);
//        sphereDraw(0,1,0,1.5f,-1);
//        sphereDraw(0,0,1,1.0f,1);
//}
//void sphereUpdate() {
//        rot++;
//}
//
//int main (int argc, const char **argv) {
//    jWorld world       = jWorld();
//        jGridBox grid = jGridBox(2.5f,2.5f,2.5f);
//        jLinkedDraw linked = jLinkedDraw(sph,sphereUpdate,0,0);
//        jOrbit orbit     = jOrbit(2,40,60,720);
//
//
//        grid.add(&linked);
//        orbit.add(&grid,-3,50,0,0);
//        world.add(&orbit);
//        world.goGL(argc,argv);
//
//    return 0;
```

```
//}
```

jglTexture.h

```
/*
 *  jglTexture.h
 *  Module
 *
 *  Created by Jon Snelling on 4/23/06.
 *  Copyright 2006 JumpMast3r. All rights reserved.
 *
 */

#include "jglIntro.h"

class jglTexture {
public:
	jglTexture();
	GLuint getImage(char *name);
private:
	#ifdef useCF
		CFMutableArrayRef images;
	#else
		popqueue images;
	#endif
	bool inited;
};
```

jglTexture.cpp

```
/*
 *  jglTexture.cpp
 *  Module
 *
 *  Created by Jon Snelling on 4/23/06.
 *  Copyright 2006 JumpMast3r. All rights reserved.
 *
 */

#include "jglTexture.h"
#include <IL/macconfig.h>
#include <IL/ilut.h>

typedef struct {
	char *name;
```

```
	GLuint addr;
} imageRecord;

jglTexture::jglTexture() {
	#ifdef useCF
		images = CFArrayCreateMutable(NULL,0,NULL);
	#else
		images = popqueueAlloc();
	#endif
	inited=false;
};

GLuint jglTexture::getImage(char *name) {
if (!inited) {
		ilutRenderer(ILUT_OPENGL);
//		ilutInit();
		ilInit();
		inited=true;
	}

	imageRecord *temp;
	#ifdef useCF
		for (int i = 0; i < CFArrayGetCount(images); i++) {
			temp = (imageRecord*)CFArrayGetValueAtIndex(images,i);
	#else
		for (int i = 0; i < images.length; i++) {
			temp = (imageRecord*)touch(&images,i);
	#endif
		if (temp->name==name)
			return temp->addr;
	}

	ILuint ImageName;
	ilGenImages(1, &ImageName);
	ilBindImage(ImageName);
	ilLoadImage(name);

	ILuint Width,Height,Depth;
	Width = ilGetInteger(IL_IMAGE_WIDTH);
	Height = ilGetInteger(IL_IMAGE_HEIGHT);
	Depth = ilGetInteger(IL_IMAGE_DEPTH);
//	printf("%d %d\n",Width,Height);
	ILuint newW,newH;
	for (newW = 1; newW < Width; newW*=2);
	for (newH = 1; newH < Width; newH*=2);
	iluScale(newW,newH,Depth);
```

```
//      iluAlienify();

        GLuint Texture;
        Texture = ilutGLBindTexImage();
        printf("%d\n",Texture);

        temp = new imageRecord;
        temp->name=name;
//      ilutGLBindTexImage();
//      ilutGLLoadImage(name);
        temp->addr=Texture;

        ilDeleteImages(1, &ImageName);

        #ifdef useCF
                CFArrayAppendValue(images,temp);
        #else
                push(&images, temp);
        #endif

        return temp->addr;
}
```

jglPicker.h

```
/*
 *  jglPicker.h
 *  Module
 *
 *  Created by Jon Snelling on 9/4/05.
 *  Copyright 2005 JumpMast3r. All rights reserved.
 *
 */

#ifndef __jon_picker__
#define __jon_picker__

#include "jglIntro.h"

int             jglStartPicking(void (*func)(),int cursorX, int cursorY, GLuint *selectBuf, int
size, float ar);
void    jglProcessHits (GLint hits, GLuint buffer[]);
int             pickedName(GLint hits, GLuint buffer[]);
#endif
```

jglPicker.cpp

```
/*
 *  jglPicker.c
 *  Module
 *
 *  Created by Jon Snelling on 9/4/05.
 *  Copyright 2005 JumpMast3r. All rights reserved.
 *
 *      Parts of this code have been modified and adapted from
 *      from freely availble code found in the "Red Book" :-)
 */

#include <stdio.h>
#include "jglPicker.h"

int jglStartPicking(void (*func)(),int cursorX, int cursorY, GLuint *selectBuf, int size, float
ar) {
        GLint viewport[4];
        glSelectBuffer(size,selectBuf);
        glRenderMode(GL_SELECT);

        glMatrixMode(GL_PROJECTION);
        glPushMatrix();
        glLoadIdentity();

        glGetIntegerv(GL_VIEWPORT,viewport);
        gluPickMatrix(cursorX,viewport[3]-cursorY,
                                        2,2,viewport);
        gluPerspective(45,ar,0.1,100);
        glMatrixMode(GL_MODELVIEW);
        glInitNames();

        int hits;

//      #ifdef DrawGLScene
        func();
//      #endif

        // restoring the original projection matrix
        glMatrixMode(GL_PROJECTION);
        glPopMatrix();
        glMatrixMode(GL_MODELVIEW);
        glFlush();
```

```
        // returning to normal rendering mode
        hits = glRenderMode(GL_RENDER);

        // if there are hits process them
//      if (hits != 0)
//                      jglProcessHits(hits,selectBuf);

        return hits;
}

void jglProcessHits (GLint hits, GLuint buffer[]) {
   unsigned int i, j;
   GLuint names, *ptr, minZ,*ptrNames=0, numberOfNames=0;

   printf ("hits = %d\n", (int)hits);
   ptr = (GLuint *) buffer;
   minZ = 0xffffffff;
   for (i = 0; i < hits; i++) {
      names = *ptr;
         ptr++;
         if (*ptr < minZ) {
                numberOfNames = names;
                minZ = *ptr;
                ptrNames = ptr+2;
         }

         ptr += names+2;
       }
  printf ("The closest hit names are ");
  ptr = ptrNames;
  for (j = 0; j < numberOfNames; j++,ptr++) {
     printf ("%d ", (int)*ptr);
  }
  printf ("\n");

}
int pickedName(GLint hits, GLuint buffer[]) {
   unsigned int i;//, j;
   GLuint names, *ptr, minZ,*ptrNames=0, numberOfNames;

   ptr = (GLuint *) buffer;
   minZ = 0xffffffff;
   for (i = 0; i < hits; i++) {
      names = *ptr;
         ptr++;
         if (*ptr < minZ) {
```

```
                numberOfNames = names;
                minZ = *ptr;
                ptrNames = ptr+2;
            }

            ptr += names+2;
        }
//  printf ("The closest hit names are ");
  ptr = ptrNames;
//  for (j = 0; j < numberOfNames; j++,ptr++) {
//     printf ("%d ", *ptr);
//  }
//  printf ("\n");
//          ptr--;
            ptr+=(numberOfNames-1);
            return ptr[0];
}
```

jglMesh.h

```
/*
 *  jglMesh.h
 *  Module
 *
 *  Created by Jon Snelling on 4/24/06.
 *  Copyright 2006 JumpMast3r. All rights reserved.
 *
 */

#include "jglIntro.h"

class jglMesh {
public:
                    jglMesh();

                    bool loadMesh(char *name);
                    bool drawMesh(char *name);
private:

#ifdef useCF
            CFMutableArrayRef meshList;
            CFMutableArrayRef vertexes;
            CFMutableArrayRef normals;
            CFMutableArrayRef faces;
#else
            popqueue meshList;
```

```
        popqueue vertexes;
        popqueue normals;
        popqueue faces;
#endif
};
```

jglMesh.cpp

```
/*
 *  jglMesh.cpp
 *  Module
 *
 *  Created by Jon Snelling on 4/24/06.
 *  Copyright 2006 JumpMast3r. All rights reserved.
 *
 */

#include "jglMesh.h"

typedef struct {
        float           x;
        float           y;
        float           z;
} jglVertex;
typedef struct {
        int             p1;
        int             p2;
        int             p3;
        int             n1;
        int             n2;
        int             n3;
} jglFace;
typedef struct {
        char *name;
        int     slot;
} meshRecord;

jglMesh::jglMesh() {
        #ifdef useCF
                meshList = CFArrayCreateMutable(NULL,0,NULL);
                vertexes = CFArrayCreateMutable(NULL,0,NULL);
                normals = CFArrayCreateMutable(NULL,0,NULL);
                faces = CFArrayCreateMutable(NULL,0,NULL);
        #else
                meshList = popqueueAlloc();
                vertexes = popqueueAlloc();
```

```
		normals = popqueueAlloc();
		faces = popqueueAlloc();
	#endif

}

bool jglMesh::drawMesh(char *name) {
	meshRecord *ptr;

	#ifdef useCF
		for (int i = 0; i < CFArrayGetCount(meshList); i++) {
			ptr = (meshRecord*)CFArrayGetValueAtIndex(meshList,i);
	#else
		for (int i = 0; i < meshList.length; i++) {
			ptr = (meshRecord*)touch(&meshList,i);
	#endif
		if (name==ptr->name) {
			glCallList(ptr->slot);
			return true;
		}
	}
	loadMesh(name);
	drawMesh(name);
	return true;
}

bool jglMesh::loadMesh(char *name) {
	int meTe;
	FILE *file = fopen(name, "rb");
   if(	file==NULL ) {
//	fclose(file);
		printf("file read error\n");
	return false;
	}

	meTe=glGenLists(1);
	jglVertex *temp = new jglVertex;
	temp->x=0;
	temp->y=0;
	temp->z=0;
	#ifdef useCF
		CFArrayAppendValue(normals,temp);
	#else
		push(&normals, temp);
	#endif
```

```
	char buffer[20];
	fscanf(file, "%s ",&buffer);
	while (fscanf(file, "%s ",&buffer)!=EOF) {
		if (buffer[0]=='v') {
			if (buffer[1]=='n') {
//				printf("found normal\n");
				jglVertex *temp = new jglVertex;
				fscanf(file, " %f %f %f\n",&temp->x,&temp->y,&temp->z);
//				printf("%f %f %f normal\n",temp->x,temp->y,temp->z);
				#ifdef useCF
					CFArrayAppendValue(normals,temp);
				#else
					push(&normals, temp);
				#endif
			} else {
				jglVertex *temp = new jglVertex;
				fscanf(file, " %f %f %f\n",&temp->x,&temp->y,&temp->z);
//				printf("%f %f %f\n",temp->x,temp->y,temp->z);
				#ifdef useCF
					CFArrayAppendValue(vertexes,temp);
				#else
					push(&vertexes, temp);
				#endif
			}
		} else if (buffer[0]=='f') {
//			printf("found face\n");
			jglFace *temp = new jglFace;
			fscanf(file, " %d//%d %d//%d %d//%d\n",&temp->p1,&temp-
>n1,&temp->p2,&temp->n2,&temp->p3,&temp->n3);
//			printf(" %d//%d %d//%d %d//%d\n",temp->p1,temp->n1,temp-
>p2,temp->n2,temp->p3,temp->n3);
			#ifdef useCF
				CFArrayAppendValue(faces,temp);
			#else
				push(&faces, temp);
			#endif
		} else {
			fscanf(file,"\n");
		}
	}


	glNewList(meTe,GL_COMPILE);
	glRotatef(-90,1,0,0);
	jglFace *ptr;
	#ifdef useCF
```

```
	for (int i = 0; i < CFArrayGetCount(faces); i++) {
		ptr = (jglFace*)CFArrayGetValueAtIndex(faces,i);
#else
	for (int i = 0; i < faces.length; i++) {
		ptr = (jglFace*)touch(&faces,i);
#endif
	glBegin(GL_TRIANGLES);
	jglVertex *tmp2 = (jglVertex*)CFArrayGetValueAtIndex(normals,ptr->n1);
	glNormal3f(tmp2->x,tmp2->y,tmp2->z);
	tmp2 = (jglVertex*)CFArrayGetValueAtIndex(vertexes,ptr->p1);
	glVertex3f(tmp2->x,tmp2->y,tmp2->z);
	tmp2 = (jglVertex*)CFArrayGetValueAtIndex(normals,ptr->n2);
	glNormal3f(tmp2->x,tmp2->y,tmp2->z);
	tmp2 = (jglVertex*)CFArrayGetValueAtIndex(vertexes,ptr->p2);
	glVertex3f(tmp2->x,tmp2->y,tmp2->z);
	tmp2 = (jglVertex*)CFArrayGetValueAtIndex(normals,ptr->n3);
	glNormal3f(tmp2->x,tmp2->y,tmp2->z);
	tmp2 = (jglVertex*)CFArrayGetValueAtIndex(vertexes,ptr->p3);
	glVertex3f(tmp2->x,tmp2->y,tmp2->z);
	glEnd();
}
glRotatef(90,1,0,0);
glEndList();

#ifdef useCF
	for (int i = 0; i < CFArrayGetCount(vertexes); i++) {
		temp = (jglVertex*)CFArrayGetValueAtIndex(vertexes,i);
#else
	for (int i = 0; i < vertexes.length; i++) {
		temp = (jglVertex*)touch(&vertexes,i);
#endif
	delete temp;
}

#ifdef useCF
	for (int i = 0; i < CFArrayGetCount(normals); i++) {
		temp = (jglVertex*)CFArrayGetValueAtIndex(normals,i);
#else
	for (int i = 0; i < normals.length; i++) {
		temp = (jglVertex*)touch(&normals,i);
#endif
	delete temp;
}

#ifdef useCF
	for (int i = 0; i < CFArrayGetCount(faces); i++) {
```

```
                ptr = (jglFace*)CFArrayGetValueAtIndex(faces,i);
        #else
                for (int i = 0; i < faces.length; i++) {
                        ptr = (jglFace*)touch(&faces,i);
        #endif
                delete ptr;
        }
        CFArrayRemoveAllValues(vertexes);
        CFArrayRemoveAllValues(normals);
        CFArrayRemoveAllValues(faces);

        meshRecord *mesh = new meshRecord;
        mesh->name = name;
        mesh->slot = meTe;
        #ifdef useCF
                CFArrayAppendValue(meshList,mesh);
        #else
                push(&meshList, mesh);
        #endif

        fclose(file);
        return true;
}
```

jglLight.h

```
/*
 *  jglLight.h
 *  Module
 *
 *  Created by Jonathan Snelling on 3/10/05.
 *  Copyright 2005 JumpMast3r. All rights reserved.
 *
 */

#ifndef __jon_light__
#define __jon_light__

#include "jglIntro.h"

//structures
typedef struct {                        // Create A Structure
        GLfloat position[4];
        GLfloat ambient[4];             // 4 values for ambient color
        GLfloat diffuse[4];
        GLfloat specular[4];
```

```
} jglLight;                                          // Structure Name

typedef struct {
        float phi;
        float theta;
        float r;
        bool on;
} jglTripod;

void jglSetjglLight(GLuint target, jglLight *jglLight);
void jglFrontLight(GLuint target);
void jglDefaultLight(GLuint target);
void jglDefaultLightAt3f(GLuint target, float x, float y, float z);
void jglDimLight(GLuint target);
void jglMoveLight(GLuint target, jglTripod *pos);
void jglMoveLightBy(GLuint target, jglTripod *pos, float phi, float theta);


void jglLightColor4f(GLuint target, float red, float green, float blue, float alpha);
void jglLightColor3f(GLuint target, float red, float blue, float green);

#endif
```

jglLight.cpp

```
/*
 *  jglLight.c
 *  Module
 *
 *  Created by Jonathan Snelling on 3/10/05.
 *  Copyright 2005 JumpMast3r. All rights reserved.
 *
 */

#include "jglLight.h"

//GLfloat jglLightPosition[] = { 0.0f, 0.0f, 2.0f, 1.0f };
//GLfloat jglLightAmbient[] = { 0.5f, 0.5f, 0.5f, 1.0f };
//GLfloat jglLightDiffuse[] = { 1.0f, 1.0f, 1.0f, 1.0f };
//GLfloat jglLightSpecular[] = { 1.0f, 1.0f, 1.0f, 1.0f };

void jglSetLight(GLuint target, jglLight *jglLight) {
        glLightfv(target, GL_POSITION, jglLight->position);
        glLightfv(target, GL_AMBIENT, jglLight->ambient);
        glLightfv(target, GL_DIFFUSE, jglLight->diffuse);
        glLightfv(target, GL_SPECULAR, jglLight->specular);
```

```
}

void jglDefaultLight(GLuint target) {
	jglLight def = {
		{ 0.0f, 0.0f, 0.0f, 1.0f },
		{ 0.0f, 0.0f, 0.0f, 1.0f },
		{ 1.0f, 1.0f, 1.0f, 1.0f },
		{ 0.5f, 0.5f, 0.5f, 0.5f }
	};
	jglSetLight(target,&def);
}

void jglDefaultLightAt3f(GLuint target, float x, float y, float z) {
	jglLight def = {
		{	x,	y,	z,	1},
		{ 0.0f, 0.0f, 0.0f, 1.0f },
		{ 1.0f, 1.0f, 1.0f, 1.0f },
		{ 0.5f, 0.5f, 0.5f, 0.5f }
	};
	jglSetLight(target,&def);
}

void jglFrontLight(GLuint target) {
	jglLight def = {
		{ -0.0f, 0.0f, 2.0f, 1.0f },
		{ 0.0f, 0.0f, 0.0f, 1.0f },
		{ 1.0f, 1.0f, 1.0f, 1.0f },
		{ 0.5f, 0.5f, 0.5f, 1.0f }
	};
	jglSetLight(target,&def);
}
void jglDimLight(GLuint target) {
	jglLight def = {
		{ -0.5f, 0.5f, 1.0f, 1.0f },
		{ 0.0f, 0.0f, 0.0f, 1.0f },
		{ 0.5f, 0.5f, 0.5f, 1.0f },
		{ 0.5f, 0.5f, 0.5f, 1.0f }
	};
	jglSetLight(target,&def);
}
void jglMoveLight(GLuint target, jglTripod* pos) {
	GLfloat p[4] = {
		pos->r*cos(pos->phi/180.0f*M_PI)*sin(pos->theta/180.0f*M_PI),
		pos->r*sin(pos->phi/180.0f*M_PI),
		pos->r*cos(pos->phi/180.0f*M_PI)*cos(pos->theta/180.0f*M_PI),
		1.0f
```

```
	};
	glLightfv(target, GL_POSITION, p);
}

void jglMoveLightBy(GLuint target, jglTripod *pos, float phi, float theta) {
	GLfloat p[4] = {
		pos->r*cos((phi+pos->phi)/180.0f*M_PI)*sin((theta+pos->theta)/
180.0f*M_PI),
		pos->r*sin((phi+pos->phi)/180.0f*M_PI),
		pos->r*cos((phi+pos->phi)/180.0f*M_PI)*cos((theta+pos->theta)/
180.0f*M_PI),
		1.0f
	};
	glLightfv(target, GL_POSITION, p);
}

void jglLightColor4f(GLuint target, float red, float green, float blue, float alpha) {
	float temp[4] = {red,green,blue,alpha};
	glLightfv(target, GL_DIFFUSE, temp);
}
void jglLightColor3f(GLuint target, float red, float green, float blue) {
	jglLightColor4f(target, red, green, blue, 1.0f);
}

//structures
//typedef struct {				// Create A Structure
//	GLfloat position[4];
//	GLfloat ambient[4];			// 4 values for ambient color
//	GLfloat diffuse[4];
//	GLfloat specular[4];
//} jglLight;
```

jglIntro.h

```
/*
 *  jglIntro.h
 *
 *  Created by Jonathan Snelling on 1/13/05.
 *  Copyright 2005 JumpMast3r. All rights reserved.
 *
 *  Just a header file which will set up basic c'isms
 *  for easy use accross systems.
 *
 */
```

```
#ifndef JON_INTRO

#if defined(__APPLE__) || defined(MACOSX)
	#include <GLUT/glut.h>
	//use apple's CoreFoundation
	#include <CoreFoundation/CoreFoundation.h>
	#define useCF
#else
	#include <GL/glut.h>
	#include <time.h>
	#include <stdio.h>
	#include <time.h>
	#include "popqueue.h"
#endif

#ifndef bool
#define bool int
#define true 1
#define false 0
#endif

#ifndef sin
	#include <math.h>
#endif

#ifndef NULL
#define NULL                            0
#endif

#ifndef M_E
#define M_E            2.7182818284590452354   /* e */
#endif

#ifndef M_LOG2E
#define M_LOG2E        1.4426950408889634074   /* log_2 e */
#endif

#ifndef M_LOG10E
#define M_LOG10E       0.43429448190325182765  /* log_10 e */
#endif

#ifndef M_LN2
#define M_LN2          0.69314718055994530942  /* log_e 2 */
#endif

#ifndef M_LN10
```

```
#define M_LN10         2.30258509299404568402  /* log_e 10 */
#endif

#ifndef M_PI
#define M_PI           3.14159265358979323846  /* pi */
#endif

#ifndef M_PI_2
#define M_PI_2         1.57079632679489661923  /* pi/2 */
#endif

#ifndef M_PI_4
#define M_PI_4         0.78539816339744830962  /* pi/4 */
#endif

#ifndef M_1_PI
#define M_1_PI         0.31830988618379067154  /* 1/pi */
#endif

#ifndef M_2_PI
#define M_2_PI         0.63661977236758134308  /* 2/pi */
#endif

#ifndef M_2_SQRTPI
#define M_2_SQRTPI     1.12837916709551257390  /* 2/sqrt(pi) */
#endif

#ifndef M_SQRT2
#define M_SQRT2        1.41421356237309504880  /* sqrt(2) */
#endif

#ifndef M_SQRT1_2
#define M_SQRT1_2      0.70710678118654752440  /* 1/sqrt(2) */
#endif

#define JON_INTRO
#endif
```

jWorld.h

```
/*
 *  jWorld.h
 *  Module
 *
 *  Created by Jon Snelling on 2/4/06.
```

```
 *  Copyright 2006 JumpMast3r. All rights reserved.
 *
 */

#ifndef JON_JWORLD
#define JON_JWORLD
#include "jglIntro.h"
#include "jObject.h"
#include "jglTexture.h"
#include "jglMesh.h"


int             fps                     = 25,
                width           = 640,
                height          = 480;
long    capabilities    = 0,
                enabled                 = 0;

class jWorld {
public:
        jWorld();
        jWorld(char *windowTitle);
        ~jWorld();
        GLvoid static Timer(int value);
        GLvoid static Keyboard(unsigned char key, int x, int y);
        GLvoid static Mouse(int button, int state, int x, int y);
        GLvoid static ReSizeGLScene(int Width, int Height);
        GLvoid static DrawGLScene(GLvoid);
        GLuint static getImage(char *name);
        bool static loadMesh(char *name);
        bool static drawMesh(char *name);
        GLvoid InitGL(GLvoid);
        int goGL (int argc, const char **argv);

        void *add(jObject *adde);
        int addClickable(jObject *adde);
        void *addWithKey(jObject *adde, char key);
        int addClickableWithKey(jObject *adde, char key);
        int static registerName(jObject *obj);
        bool static registerKey(jObject *obj,char key);
private:
        static jObject obj;
        static jglTexture textures;
        static jglMesh meshes;
        #ifdef useCF
                static CFMutableArrayRef clickList;
```

```
		static CFMutableArrayRef keyList;
	#else
		static popqueue clickList;
		static popqueue keyList;
	#endif
	static int names;
	static char *name;
};
#endif
```

jWorld.cpp

```
/*
 *  jWorld.cpp
 *  Module
 *
 *  Created by Jon Snelling on 2/4/06.
 *  Copyright 2006 JumpMast3r. All rights reserved.
 *
 */

#include "jWorld.h"
#include "jObject.h"
#include "jglPicker.h"
#include "jglLight.h"
#include "jglFunctionCaller.h"

jObject jWorld::obj;
jglTexture jWorld::textures;
jglMesh jWorld::meshes;
int jWorld::names;
#ifdef useCF
	CFMutableArrayRef jWorld::clickList;
	CFMutableArrayRef jWorld::keyList;
#else
	popqueue jWorld::clickList;
	popqueue jWorld::keyList;
#endif

typedef struct {
	jObject* object;
	int name;
} nameLib;

typedef struct {
	jObject* object;
```

```
	char key;
} keyLib;

jWorld::jWorld() {
	#ifdef useCF
		clickList = CFArrayCreateMutable(NULL,0,NULL);
		keyList = CFArrayCreateMutable(NULL,0,NULL);
	#else
		clickList = popqueueAlloc();
		keyList = popqueueAlloc();
	#endif
}

jWorld::~jWorld() {
	nameLib *ptr;

	#ifdef useCF
		for (int i = 0; i < CFArrayGetCount(clickList); i++) {
			ptr = (nameLib*)CFArrayGetValueAtIndex(clickList,i);
	#else
		for (int i = 0; i < clickList.length; i++) {
			ptr = (nameLib*)touch(&clickList,i);
	#endif
		delete ptr;
	}

}

GLuint jWorld::getImage(char *name) {
	return textures.getImage(name);
}

bool jWorld::loadMesh(char *name) {
	return meshes.loadMesh(name);
}
bool jWorld::drawMesh(char *name) {
	return meshes.drawMesh(name);
}

GLvoid jWorld::Timer(int value) {
	obj.update();
   glutPostRedisplay();
	glutTimerFunc((int)(1000.0f/(float)fps),Timer,(int)(1000.0f/(float)fps));
}

GLvoid jWorld::Keyboard(unsigned char key, int x, int y) {
```

```
#pragma unused (x, y)
	keyLib *ptr;
	#ifdef useCF
		for (int i = 0; i < CFArrayGetCount(clickList); i++) {
			ptr = (keyLib*)CFArrayGetValueAtIndex(clickList,i);
	#else
		for (int i = 0; i < clickList.length; i++) {
			ptr = (keyLib*)touch(&clickList,i);
	#endif
		if (ptr->key==key)
			ptr->object->keypressNotify(key);
	}

   glutPostRedisplay();
}

/*	Mouse
 *
 *	deals with mouse click
 */
GLvoid jWorld::Mouse(int button, int state, int x, int y)  {
	if (state == GLUT_DOWN) {
		GLuint buffer[512];
		int hits,name;
		hits = jglStartPicking(DrawGLScene, x,y,buffer,512, (float)width/(float)
height);
		if (hits!=0) {
			name = pickedName(hits,buffer);
			nameLib *ptr;
			#ifdef useCF
				for (int i = 0; i < CFArrayGetCount(clickList); i++) {
					ptr = (nameLib*)CFArrayGetValueAtIndex(clickList,i);
			#else
				for (int i = 0; i < clickList.length; i++) {
					ptr = (nameLib*)touch(&clickList,i);
			#endif
				if (ptr->name==name)
					ptr->object->clickedObject(name);
			}
		}
	}
}

GLvoid jWorld::ReSizeGLScene(int Width, int Height) {
   glViewport (0, 0, (GLsizei) Width, (GLsizei) Height);
   glMatrixMode(GL_PROJECTION);
```

```
    glLoadIdentity();
//      int j = .1;

    gluPerspective(45.0,
                (GLfloat) Width / (GLfloat) Height,
                0.1,
                100.0);

        width=Width;
        height=Height;

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

GLvoid jWorld::InitGL(GLvoid) {
        glClearColor(0.0f, 0.0f, 0.0f, 0.5f);               // This Will Clear The Background
Color To Black
        glClearDepth(1.0);                                  // Enables Clearing Of The Depth
Buffer
        glDepthFunc(GL_LESS);                               // The Type Of Depth Test To Do
        glEnable(GL_DEPTH_TEST);                            // Enables Depth Testing
        glShadeModel(GL_SMOOTH);                            // Enables Smooth Color
Shading
        glMatrixMode(GL_PROJECTION);
        glBlendFunc(GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALPHA);
        glEnable(GL_BLEND);
        glLoadIdentity();                                   // Reset The Projection Matrix
        gluPerspective(45.0f,
                                (GLfloat)width/(GLfloat)height,
                                0.1f,
                                100.0f);                    // Calculate The Aspect Ratio
Of The Window
        glMatrixMode(GL_MODELVIEW);


        float lmodel_ambient[] = {0.4, 0.4, 0.4, 1.0};
        float lmodel_localviewer[] = {0.0};
        jglFrontLight(GL_LIGHT0);
//      jglDefaultLight(GL_LIGHT0);
        glLightModelfv(GL_LIGHT_MODEL_AMBIENT, lmodel_ambient);
        glLightModelfv(GL_LIGHT_MODEL_LOCAL_VIEWER, lmodel_localviewer);
        glEnable(GL_LIGHTING);
        glEnable(GL_LIGHT0);
```

```
        glInitNames();
}

GLvoid jWorld::DrawGLScene(GLvoid) {
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   glLoadIdentity();

        glTranslatef(0,0,-10);
        obj.draw();

        glutSwapBuffers();
   glFlush();
}

int jWorld::goGL (int argc, const char **argv) {
        glutInit(&argc, (char **)argv);
        glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
        glutInitWindowSize (width, height);
        glutInitWindowPosition (100, 100);
        glutCreateWindow ("CHANGE THIS");

        InitGL();

        glutDisplayFunc(DrawGLScene);
        glutReshapeFunc(ReSizeGLScene);
        glutKeyboardFunc(Keyboard);
        glutMouseFunc(Mouse);
        glutTimerFunc((int)(1000.0f/(float)fps),Timer,(int)(1000.0f/(float)fps));

        glutMainLoop();

        return 0;
}

void *jWorld::add(jObject *adde) {
        return obj.add(adde);
}

int jWorld::addClickable(jObject *adde) {
        add(adde);
        return registerName(adde);
}
void *jWorld::addWithKey(jObject *adde, char key) {
```

```
	add(adde);
	registerKey(adde,key);
	return adde;
}
int jWorld::addClickableWithKey(jObject *adde, char key) {
	add(adde);
	registerKey(adde,key);
	return registerName(adde);
}

int jWorld::registerName(jObject *obj) {
	nameLib *temp = new nameLib;
	temp->object = obj;
	temp->name = names++;
	#ifdef useCF
		CFArrayAppendValue(clickList,temp);
	#else
		push(&clickList, temp);
	#endif
	obj->setName(temp->name);
	return temp->name;
}

bool jWorld::registerKey(jObject *obj,char key) {
	keyLib *temp = new keyLib;
	temp->object = obj;
	temp->key = key;
	#ifdef useCF
		CFArrayAppendValue(clickList,temp);
	#else
		push(&clickList, temp);
	#endif
	return true;
}
```

jSprite.h

```
/*
 *  jSprite.h
 *  Module
 *
 *  Created by Jon Snelling on 4/23/06.
 *  Copyright 2006 JumpMast3r. All rights reserved.
 *
 */
```

```
#include "jObject.h"

class jSprite: public jObject {
public:
	jSprite(char *name);
	jSprite(char *name, float Size);

	void drawObject();
	void updateObject();

	void setSize(float newSize);
	void clickedObject(int name);
	void jSprite::keypressNotify(char key);


	void setColor(float r, float g, float b);

	int		rot;
	float	size;
	float	rC,gC,bC;
	char *image;
};
```

jSprite.cpp

```
/*
 *  jSprite.cpp
 *  Module
 *
 *  Created by Jon Snelling on 4/23/06.
 *  Copyright 2006 JumpMast3r. All rights reserved.
 *
 */

#include "jSprite.h"
#include "jWorld.h"

jSprite::jSprite(char *name) {
	rot		= 0;
	size	= 1.0f;
	rC = gC = bC = 1;
	image = name;
}
jSprite::jSprite(char *name, float Size) {
	rot		= 0;
	size	= Size;
```

```
	image = name;
}

void jSprite::setSize(float newSize) {
	size = newSize;
}

void jSprite::drawObject() {
	GLuint temp = jWorld::getImage(image);
//	printf("%d",temp);
	glColor3f(rC,gC,bC);
	glRotatef(rot,1,0,0);
	glBindTexture(GL_TEXTURE_2D,temp);
	glBegin(GL_QUADS);
	glTexCoord2f(0,0);
	glVertex3f(-size/2,-size/2,0);
	glTexCoord2f(1,0);
	glVertex3f(size/2,-size/2,0);
	glTexCoord2f(1,1);
	glVertex3f(size/2,size/2,0);
	glTexCoord2f(0,1);
	glVertex3f(-size/2,size/2,0);
	glEnd();
	glBindTexture(GL_TEXTURE_2D,0);
}
void jSprite::updateObject() {
	if (rot>0) rot-=10;
}

void jSprite::clickedObject(int name) {
	rot+=360;
}

void jSprite::keypressNotify(char key) {
	rot+=180;
}

void jSprite::setColor(float r, float g, float b) {
	rC = r;
	gC = g;
	bC = b;
}
```

jSpinner.h

```
/*
```

```
 *  jSpinner.h
 *  Module
 *
 *  Created by Jon Snelling on 4/22/06.
 *  Copyright 2006 JumpMast3r. All rights reserved.
 *
 */

#include "jObject.h"
class jSpinner: public jObject {
public:
	jSpinner();
	jSpinner(bool setSolid);
	jSpinner(float Size);

	void drawObject();
	void updateObject();

	void setSize(float newSize);
	void clickedObject(int name);
	void jSpinner::keypressNotify(char key);


	void setColor(float r, float g, float b);

	int		rot;
	float	size;
	float	rC,gC,bC;
};
```

jSpinner.cpp

```
/*
 *  jSpinner.cpp
 *  Module
 *
 *  Created by Jon Snelling on 4/22/06.
 *  Copyright 2006 JumpMast3r. All rights reserved.
 *
 */

#include "jSpinner.h"

jSpinner::jSpinner() {
	rot		= 0;
	size	= 1.0f;
```

```
	rC = gC = bC = 1;
}
jSpinner::jSpinner(float Size) {
	rot		= 0;
	size	= Size;
}

void jSpinner::setSize(float newSize) {
	size = newSize;
}

void jSpinner::drawObject() {
	glColor3f(rC,gC,bC);
	glRotatef(rot,1,0,0);
	glBegin(GL_QUADS);
	glVertex3f(-size/2,-size/2,0);
	glVertex3f(size/2,-size/2,0);
	glVertex3f(size/2,size/2,0);
	glVertex3f(-size/2,size/2,0);
	glEnd();
}
void jSpinner::updateObject() {
	if (rot>0) rot-=10;
}

void jSpinner::clickedObject(int name) {
	rot+=360;
}

void jSpinner::keypressNotify(char key) {
	rot+=180;
}

void jSpinner::setColor(float r, float g, float b) {
	rC = r;
	gC = g;
	bC = b;
}
```

jSphere.h

```
/*
 *  jObject.cpp
 *  Module
 *
 *  Created by Jon Snelling on 2/4/06.
```

```
 *  Copyright 2006 JumpMast3r. All rights reserved.
 *
 */

#include "jObject.h"
class jSphere: public jObject {
public:
	jSphere();
	jSphere(bool setSolid);
	jSphere(float Size);
	jSphere(float Size,bool setSolid);

	void drawObject();
	void updateObject();

	void setSolid(bool setSolid);
	void setSize(float newSize);

	int		rot;
	bool	solid;
	float	size;
};
```

jSphere.cpp

```
/*
 *  jObject.cpp
 *  Module
 *
 *	the jSphere object is an example class which demonstrates
 *	the simplicity of encapsulating the glut sphere and rotate
 *	it in time.
 *
 *  Created by Jon Snelling on 2/4/06.
 *  Copyright 2006 JumpMast3r. All rights reserved.
 *
 */
#include <iostream>
using std::cout;
using std::endl;
#include "jSphere.h"

jSphere::jSphere() {
	rot		= 0;
	solid	= false;
	size	= 1.0f;
```

```
}
jSphere::jSphere(bool setSolid) {
	rot		= 0;
	size	= 1.0f;
	solid	= setSolid;
}
jSphere::jSphere(float Size) {
	rot		= 0;
	solid	= false;
	size	= Size;
}
jSphere::jSphere(float Size, bool setSolid) {
	rot		= 0;
	size	= Size;
	solid	= setSolid;
}

void jSphere::setSize(float newSize) {
	size = newSize;
}

void jSphere::drawObject() {
	glRotatef(rot,1,0,0);
	if (solid) glutSolidSphere(size,20,20);
	else glutWireSphere(size,20,20);
}
void jSphere::updateObject() {
	rot++;
}

void jSphere::setSolid(bool setSolid) {
	solid = setSolid;
}
```

jOrbit.h

```
/*
 *  jOrbit.h
 *  Module
 *
 *	the jOrbit module is a demonstration of a situation
 *	where you might want to overload the standard object
 *	handling method of an object. Specificially the the
 *	jOrbit expands on the idea of the jRotator and allows
 *	for a number of objects to rotate around the same axis
 *	but with different offsets (in spherical notation)
```

```
 *       from the origin.
 *
 *  Created by Jonathan Snelling on 4/17/06.
 *  Copyright 2006 JumpMast3r. All rights reserved.
 *
 */

#include "jObject.h"

class jOrbit: public jObject {
public:
        jOrbit();
        jOrbit(int Roe, int Phi, int The, int Speed);
        ~jOrbit();

        virtual void draw();
        virtual void update();
        virtual void updateObject();
        virtual void *add(jObject *adde);
                        void *add(jObject *adde, float Roe, float Phi, float The, float Spe);

        int rot;        //time marker
        int roe;                //distance in the spherical coordinate system reference
        int phi;        //first part of the spherical coordinate system reference
        int theta;              //other part of the spherical coordinate system reference
        int speed;              //increments per update cycle
};
```

jOrbit.cpp

```
/*
 *  jOrbit.cpp
 *  Module
 *
 *       the jOrbit module is a demonstration of a situation
 *       where you might want to overload the standard object
 *       handling method of an object. Specificially the the
 *       jOrbit expands on the idea of the jRotator and allows
 *       for a number of objects to rotate around the same axis
 *       but with different offsets (in spherical notation)
 *       from the origin.
 *
 *  Created by Jonathan Snelling on 4/17/06.
 *  Copyright 2006 JumpMast3r. All rights reserved.
 *
 */
```

```
#include "jOrbit.h"
#include "popqueue.h"

typedef struct {
	jObject *adde;
	float roe;
	float phi;
	float theta;
	float speed;
} fixation;

jOrbit::jOrbit() {
	rot		= 0;
	roe		= 0;
	phi		= 0;
	theta	= 0;
	speed = 0;

	#ifdef useCF
		drawList = CFArrayCreateMutable(NULL,0,NULL);
	#else
		drawList = popqueueAlloc();
	#endif
}

jOrbit::jOrbit(int Roe, int Phi, int The, int Speed) {
	jOrbit();
	roe		= Roe;
	phi		= Phi;
	theta	= The;
	speed = Speed;
}

jOrbit::~jOrbit() {
	fixation *ptr;

	#ifdef useCF
		for (int i = 0; i < CFArrayGetCount(drawList); i++) {
			ptr = (fixation*)CFArrayGetValueAtIndex(drawList,i);
	#else
		for (int i = 0; i < drawList.length; i++) {
			ptr = (fixation*)touch(&drawList,i);
	#endif
		delete ptr;
	}
```

```
}

void jOrbit::draw() {
	fixation *ptr;

	glRotatef(phi*rot*speed,0,0,1);
	glRotatef(theta*rot*speed,0,1,0);
	glTranslatef(roe,0,0);

	#ifdef useCF
		for (int i = 0; i < CFArrayGetCount(drawList); i++) {
			ptr = ((fixation*)CFArrayGetValueAtIndex(drawList,i));
	#else
		for (int i = 0; i < drawList.length; i++) {
			ptr = ((fixation*)touch(&drawList,i));
	#endif
		glRotatef(ptr->phi*rot*ptr->speed,0,0,1);
		glRotatef(ptr->theta*rot*ptr->speed,0,1,0);
		glTranslatef(ptr->roe,0,0);

		ptr->adde->draw();

		glTranslatef(-ptr->roe,0,0);
		glRotatef(ptr->theta*rot*ptr->speed,0,-1,0);
		glRotatef(ptr->phi*rot*ptr->speed,0,0,-1);
	}
	drawObject();

	glTranslatef(-roe,0,0);
	glRotatef(theta*rot*speed,0,-1,0);
	glRotatef(phi*rot*speed,0,0,-1);

}

void jOrbit::update() {
	jObject *ptr;

	#ifdef useCF
		for (int i = 0; i < CFArrayGetCount(drawList); i++) {
			ptr = (jObject*)((fixation*)CFArrayGetValueAtIndex(drawList,i))-
>adde;
	#else
		for (int i = 0; i < drawList.length; i++) {
			ptr = (jObject*)((fixation*)touch(&drawList,i))->adde;
	#endif
```

```
                ptr->update();
        }

        updateObject();

        return;
}

void jOrbit::updateObject() {
        rot++;
}


void *jOrbit::add(jObject *adde) {
        fixation *test = (fixation*)malloc(sizeof(fixation));
        test->adde    = adde;
        test->phi     = 0;
        test->roe     = 0;
        test->theta   = 0;
        test->speed = 0;
        #ifdef useCF
                CFArrayAppendValue(drawList,test);
        #else
                push(&drawList, test);
        #endif
        return adde;
}

void *jOrbit::add(jObject *adde, float Roe, float Phi, float The, float Spe) {

        fixation *test = (fixation*)malloc(sizeof(fixation));
        test->adde    = adde;
        test->phi     = Phi;
        test->roe     = Roe;
        test->theta   = The;
        test->speed = Spe;
                #ifdef useCF
                CFArrayAppendValue(drawList,test);
        #else
                push(&drawList, test);
        #endif
        return adde;
}
```

jObject.h

```
/*
 *  jObject.h
 *  Module
 *
 *  Created by Jon Snelling on 2/4/06.
 *  Copyright 2006 JumpMast3r. All rights reserved.
 *
 */

#ifndef J_OBJECT
#define J_OBJECT

#include "jglIntro.h"

class jObject {
public:
	jObject();
	~jObject();

	virtual void draw();
	virtual void drawObject();

	virtual void update();
	virtual void updateObject();

	virtual void clickedObject(int name);
	virtual void keypressNotify(char key);

	virtual void *add(jObject *adde);

	//visibility controls
	virtual bool isVisible();
			void showMe();
			void hideMe();
			void show();
			void hide();
	virtual void setName(int name);

	void setClickDelegate(void (*clickFunction)(int name));
protected:
	#ifdef useCF
		CFMutableArrayRef drawList;
	#else
		popqueue drawList;
	#endif
```

```
	int myName;
	bool visible;
	float	rC,gC,bC;

	void (*extClick)(int name);
};
#endif
```

jObject.cpp

```
/*
 *  jObject.cpp
 *  Module
 *
 *  Created by Jon Snelling on 2/4/06.
 *  Copyright 2006 JumpMast3r. All rights reserved.
 *
 */

#include "jObject.h"
#include <iostream>
	using std::cout;
	using std::endl;
#include "popqueue.h"
#include <typeinfo>

jObject::jObject() {
	#ifdef useCF
		drawList = CFArrayCreateMutable(NULL,0,NULL);
	#else
		drawList = popqueueAlloc();
	#endif
	extClick=0;
	myName = -1;
	visible = true;
	rC = gC = bC = 1;
}

jObject::~jObject() {
//	jObject *ptr;
//
//	#ifdef useCF
//		for (int i = 0; i < CFArrayGetCount(drawList); i++) {
//			ptr = (jObject*)CFArrayGetValueAtIndex(drawList,i);
//	#else
//		for (int i = 0; i < drawList.length; i++) {
```

```
//                      ptr = (jObject*)touch(&drawList,i);
//      #endif
//              delete ptr;
//      }
}

void jObject::draw() {
        jObject *ptr;

        #ifdef useCF
                for (int i = 0; i < CFArrayGetCount(drawList); i++) {
                        ptr = (jObject*)CFArrayGetValueAtIndex(drawList,i);
        #else
                for (int i = 0; i < drawList.length; i++) {
                        ptr = (jObject*)touch(&drawList,i);
        #endif
                if (ptr->isVisible())
                        ptr->draw();
        }
        if (visible) {
                glPushMatrix();
                if (myName!=-1)
                        glPushName(myName);
                glColor3f(rC,bC,gC);
                drawObject();
                if (myName!=-1)
                        glPopName();
                glPopMatrix();
        }
}

void jObject::showMe() {
        visible = true;
}
void jObject::hideMe() {
        visible = false;
}
void jObject::show() {
        jObject *ptr;
        #ifdef useCF
                for (int i = 0; i < CFArrayGetCount(drawList); i++) {
                        ptr = (jObject*)CFArrayGetValueAtIndex(drawList,i);
        #else
                for (int i = 0; i < drawList.length; i++) {
                        ptr = (jObject*)touch(&drawList,i);
        #endif
```

```
                ptr->show();
        }
        showMe();
}
void jObject::hide() {
        jObject *ptr;
        #ifdef useCF
                for (int i = 0; i < CFArrayGetCount(drawList); i++) {
                        ptr = (jObject*)CFArrayGetValueAtIndex(drawList,i);
        #else
                for (int i = 0; i < drawList.length; i++) {
                        ptr = (jObject*)touch(&drawList,i);
        #endif
                ptr->hide();
        }
        hideMe();
}

bool jObject::isVisible() {
        return visible;
}

void jObject::update() {
        jObject *ptr;
        #ifdef useCF
                for (int i = 0; i < CFArrayGetCount(drawList); i++) {
                        ptr = (jObject*)CFArrayGetValueAtIndex(drawList,i);
        #else
                for (int i = 0; i < drawList.length; i++) {
                        ptr = (jObject*)touch(&drawList,i);
        #endif
                ptr->update();
        }
        updateObject();
}

void jObject::drawObject() {
        return;
}

void jObject::updateObject() {
        return;
}

void jObject::keypressNotify(char key) {
        return;
```

```
}

void jObject::setClickDelegate(void (*clickFunction)(int name)) {
	extClick = clickFunction;
}
void jObject::clickedObject(int name) {
	if (extClick!=0) extClick(name);
	return;
}

void jObject::setName(int name) {
	myName = name;
}

void *jObject::add(jObject *adde) {
	#ifdef useCF
		CFArrayAppendValue(drawList,adde);
	#else
		push(&drawList, adde);
	#endif
	return adde;
}
```

jMotionQueue.h

```
/*
 *  jMotionQueue.h
 *  Module
 *
 *  Created by Jon Snelling on 4/23/06.
 *  Copyright 2006 JumpMast3r. All rights reserved.
 *
 */

#include "jObject.h"

typedef struct {
	float		xPos;
	float		yPos;
	float		zPos;
	float		xRot;
	float		yRot;
	float		zRot;
	float		duration;
} keyframe;
```

```
class jMotionQueue: public jObject {
public:
	jMotionQueue();
	jMotionQueue(float xpos, float ypos, float zpos);
	jMotionQueue(float xpos, float ypos, float zpos, float xrot, float yrot, float zrot);

	void moveAbsolute(float xpos, float ypos, float zpos, float xrot, float yrot, float
zrot, float duration);
	void moveRelative(float xpos, float ypos, float zpos, float xrot, float yrot, float zrot,
float duration);
	void pause(float duration);

	void draw();
	void update();
private:
	void		shiftLeft();
	void		computeSlopes();
	void		smallStep();

	float		xPos;
	float		yPos;
	float		zPos;
	float		xRot;
	float		yRot;
	float		zRot;
	float		xPosS;
	float		yPosS;
	float		zPosS;
	float		xRotS;
	float		yRotS;
	float		zRotS;
	float		timeCode;
	float		target;

	#ifdef useCF
		CFMutableArrayRef timeLine;
	#else
		popqueue timeLine;
	#endif
};
```

jMotionQueue.cpp

```
/*
 *  jMotionQueue.cpp
 *  Module
```

```
 *
 *  Created by Jon Snelling on 4/23/06.
 *  Copyright 2006 JumpMast3r. All rights reserved.
 *
 */

#include "jMotionQueue.h"

jMotionQueue::jMotionQueue(): jObject() {
	#ifdef useCF
		timeLine = CFArrayCreateMutable(NULL,0,NULL);
	#else
		timeLine = popqueueAlloc();
	#endif

	timeCode = 0;
	moveAbsolute(0,0,0,0,0,0,1);
	computeSlopes();
}
jMotionQueue::jMotionQueue(float xpos, float ypos, float zpos) {
	#ifdef useCF
		timeLine = CFArrayCreateMutable(NULL,0,NULL);
	#else
		timeLine = popqueueAlloc();
	#endif

	timeCode = 0;
	moveAbsolute(xpos, ypos, zpos, 0,0,0,1);
	computeSlopes();
}

jMotionQueue::jMotionQueue(float xpos, float ypos, float zpos, float xrot, float yrot, float
zrot) {
	#ifdef useCF
		timeLine = CFArrayCreateMutable(NULL,0,NULL);
	#else
		timeLine = popqueueAlloc();
	#endif

	timeCode = 0;
	moveAbsolute(xpos, ypos, zpos, xrot, yrot, zrot,1);
	computeSlopes();
}

void jMotionQueue::moveAbsolute(float xpos, float ypos, float zpos, float xrot, float yrot,
float zrot, float duration) {
```

```
	keyframe *sn = new keyframe;

	sn->xPos = xpos;	sn->yPos = ypos;	sn->zPos = zpos;
	sn->xRot = xrot;	sn->yRot = yrot;	sn->zRot = zrot;
	sn->duration = duration;

	#ifdef useCF
		CFArrayAppendValue(timeLine,sn);
	#else
		push(&timeLine, sn);
	#endif
}

void jMotionQueue::moveRelative(float xpos, float ypos, float zpos, float xrot, float yrot,
float zrot, float duration) {
	keyframe *sn = new keyframe;
	keyframe *ptr;

	ptr = (keyframe*)CFArrayGetValueAtIndex(timeLine,CFArrayGetCount
(timeLine)-1);

	sn->xPos = ptr->xPos + xpos;	sn->yPos = ptr->yPos + ypos;	sn->zPos =
ptr->zPos + zpos;
	sn->xRot = ptr->xRot + xrot;	sn->yRot = ptr->yRot + yrot;	sn->zRot =
ptr->zRot + zrot;
	sn->duration = duration;

	#ifdef useCF
		CFArrayAppendValue(timeLine,sn);
	#else
		push(&timeLine, sn);
	#endif
}

void jMotionQueue::pause(float duration) {
	moveRelative(0,0,0,0,0,0,duration);
}

void jMotionQueue::shiftLeft() {
	timeCode=0;
	CFArrayRemoveValueAtIndex(timeLine,0);
}
void jMotionQueue::computeSlopes() {
	keyframe *ptr1,*ptr2;

	ptr1 = (keyframe*)CFArrayGetValueAtIndex(timeLine,0);
```

```
		timeCode=0;

		xPos = ptr1->xPos;
		yPos = ptr1->yPos;
		zPos = ptr1->zPos;
		xRot = ptr1->xRot;
		yRot = ptr1->yRot;
		zRot = ptr1->zRot;

		if (CFArrayGetCount(timeLine)>1) {
			ptr2 = (keyframe*)CFArrayGetValueAtIndex(timeLine,1);

			target = ptr2->duration;

			xPosS = (ptr2->xPos - ptr1->xPos)/target;
			yPosS = (ptr2->yPos - ptr1->yPos)/target;
			zPosS = (ptr2->zPos - ptr1->zPos)/target;
			xRotS = (ptr2->xRot - ptr1->xRot)/target;
			yRotS = (ptr2->yRot - ptr1->yRot)/target;
			zRotS = (ptr2->zRot - ptr1->zRot)/target;
		}

}
void jMotionQueue::smallStep() {
		xPos+=xPosS;
		yPos+=yPosS;
		zPos+=zPosS;
		xRot+=xRotS;
		yRot+=yRotS;
		zRot+=zRotS;
		timeCode++;
}

void jMotionQueue::draw() {
//		jObject *ptr;

		glTranslatef(xPos,yPos,zPos);
		glRotatef(xRot,1,0,0);
		glRotatef(yRot,0,1,0);
		glRotatef(zRot,0,0,1);

		if (myName!=-1)
			glPushName(myName);
		jObject::draw();
		if (myName!=-1)
```

```
                glPopName();

        glRotatef(zRot,0,0,-1);
        glRotatef(yRot,0,-1,0);
        glRotatef(xRot,-1,0,0);
        glTranslatef(-xPos,-yPos,-zPos);
}

void jMotionQueue::update() {
        jObject::update();
        if (CFArrayGetCount(timeLine)==1) {
                return;
        } else {
                if (timeCode==0) {
                        computeSlopes();
                }
                smallStep();
                if (timeCode==target) {
                        shiftLeft();
                }
        }
}
```

jMesh.h

```
/*
 *  jMesh.h
 *  Module
 *
 *  Created by Jon Snelling on 4/24/06.
 *  Copyright 2006 JumpMast3r. All rights reserved.
 *
 */

#include "jObject.h"
class jMesh: public jObject {
public:
        jMesh(char *name);
        jMesh(char *name, float Scale);

        void drawObject();

        char *meshName;
        float scale;
};
```

jMesh.cpp

```
/*
 *  jMesh.cpp
 *  Module
 *
 *  Created by Jon Snelling on 4/24/06.
 *  Copyright 2006 JumpMast3r. All rights reserved.
 *
 */

#include "jMesh.h"
#include "jWorld.h"

jMesh::jMesh(char *name) {
	meshName = name;
	scale=1;
}
jMesh::jMesh(char *name, float Scale) {
	meshName = name;
	scale=Scale;
}

void jMesh::drawObject() {
	glScalef(scale,scale,scale);
//	glTranslatef(0.4,0,0);
//	jWorld::drawMesh(meshName);
//	glTranslatef(0,-0.01,0);
//	glColor3f(0,0,0);
	jWorld::drawMesh(meshName);
}
```

jLinkedDraw.h

```
/*
 *  jLinkedDraw.h
 *  Module
 *
 *  Created by Jonathan Snelling on 4/18/06.
 *  Copyright 2006 JumpMast3r. All rights reserved.
 *
 */

#include "jObject.h"

class jLinkedDraw: public jObject {
```

```
public:
	jLinkedDraw(void (*externalDraw)(),void (*externalUpdate)(), void (*externalKey)
(char key), void (*externalClick)(int name));
	void drawObject();
	void updateObject();
	void clickedObject(int name);
	void keypressNotify(char key);
private:
	void (*extDraw)();
	void (*extUpdate)();
	void (*extClick)(int name);
	void (*extKey)(char key);
};
```

jLinkedDraw.cpp

```
/*
 *  jLinkedDraw.cpp
 *  Module
 *
 *  Created by Jonathan Snelling on 4/18/06.
 *  Copyright 2006 JumpMast3r. All rights reserved.
 *
 */

#include "jLinkedDraw.h"

jLinkedDraw::jLinkedDraw(void (*externalDraw)(),void (*externalUpdate)(), void
(*externalKey)(char key), void (*externalClick)(int name)) {
	extDraw = externalDraw;
	extUpdate = externalUpdate;
	extKey = externalKey;
	extClick = externalClick;

}

void jLinkedDraw::drawObject() {
//	if (extClick!=0)
//		glPushName(myName);
	if (extDraw != 0)
		extDraw();
//	if (extClick!=0)
//		glPopName();

}
```

```
void jLinkedDraw::updateObject() {
	if (extUpdate != 0)
		extUpdate();
}

void jLinkedDraw::clickedObject(int name) {
	if (extClick != 0)
		extClick(name);
}

void jLinkedDraw::keypressNotify(char key) {
	if (extKey != 0)
		extKey(key);
}
```

jGroup.h

```
/*
 *  jGroup.h
 *  Module
 *
 *  Created by Jon Snelling on 4/23/06.
 *  Copyright 2006 JumpMast3r. All rights reserved.
 *
 */

#include "jObject.h"
class jGroup: public jObject {
public:
	jGroup();

	void	keypressNotify(char key);

	bool	isActive();
	bool	toggle();

	void	draw();
	void	update();

	bool	active;
};
```

jGroup.cpp

```
/*
 *  jGroup.cpp
```

```
 *  Module
 *
 *  Created by Jon Snelling on 4/23/06.
 *  Copyright 2006 JumpMast3r. All rights reserved.
 *
 */

#include "jGroup.h"

jGroup::jGroup() {
	active = true;
}

void jGroup::keypressNotify(char key) {
	toggle();
}

bool jGroup::isActive() {
	return active;
}
bool jGroup::toggle() {
	if (active) active=false;
	else active=true;
	return active;
}

void jGroup::draw() {
	if (active)
		jObject::draw();
}

void jGroup::update() {
	if (active)
		jObject::update();
}
```

jGridBox.h

```
/*
 *  jGridBox.h
 *  Module
 *
 *  Created by Jonathan Snelling on 4/18/06.
 *  Copyright 2006 JumpMast3r. All rights reserved.
 *
 */
```

```
#include "jObject.h"

class jGridBox: public jObject {
public:
	jGridBox();
	jGridBox(float x, float y, float z);

	void drawObject();
private:
	float xRange, yRange, zRange;
};
```

jGridBox.cpp

```
/*
 *  jGridBox.cpp
 *  Module
 *
 *  Created by Jonathan Snelling on 4/18/06.
 *  Copyright 2006 JumpMast3r. All rights reserved.
 *
 */

#include "jGridBox.h"

jGridBox::jGridBox(): jObject() {
	xRange = 1;
	yRange = 1;
	zRange = 1;
}

jGridBox::jGridBox(float x, float y, float z): jObject() {
	xRange = x;
	yRange = y;
	zRange = z;
}

void jGridBox::drawObject() {
	glColor4f(0,1,0,0.25f);
	glBegin(GL_QUADS);
	glVertex3f(0,yRange,zRange);
	glVertex3f(0,yRange,-zRange);
	glVertex3f(0,-yRange,-zRange);
	glVertex3f(0,-yRange,zRange);
```

```
        glVertex3f(-xRange,-yRange,0);
        glVertex3f(-xRange,yRange,0);
        glVertex3f(xRange,yRange,0);
        glVertex3f(xRange,-yRange,0);

        glVertex3f(-xRange,0,-yRange);
        glVertex3f(xRange,0,-yRange);
        glVertex3f(xRange,0,yRange);
        glVertex3f(-xRange,0,yRange);
        glEnd();
        glColor4f(0,0,1,1);
        glBegin(GL_LINES);
        glVertex3f(-xRange,0,0);
        glVertex3f(-xRange,0,0);
        glVertex3f(0,-yRange,0);
        glVertex3f(0,yRange,0);
        glVertex3f(0,0,-zRange);
        glVertex3f(0,0,zRange);
        glEnd();
}
```

jCartesian.h

```
/*
 *  jCartesian.h
 *  Module
 *
 *  Created by Jon Snelling on 4/22/06.
 *  Copyright 2006 JumpMast3r. All rights reserved.
 *
 */

#include "jObject.h"

class jCartesian: public jObject {
public:
        void draw();
        void update();
        void *add(jObject *adde);
        void *jCartesian::add(jObject *adde, float X, float Y, float Z);
        void *jCartesian::add(jObject *adde, float X, float Y, float Z, float Rx, float Ry, float
Rz);
};
```

jCartesian.cpp

```
/*
 *  jCartesian.cpp
 *  Module
 *
 *  Created by Jon Snelling on 4/22/06.
 *  Copyright 2006 JumpMast3r. All rights reserved.
 *
 */

#include "jCartesian.h"

typedef struct {
	jObject *obj;
	float	xT,
			yT,
			zT,
			xR,
			yR,
			zR;
} disp;

void jCartesian::draw() {
	disp *ptr;

	#ifdef useCF
		for (int i = 0; i < CFArrayGetCount(drawList); i++) {
			ptr = ((disp*)CFArrayGetValueAtIndex(drawList,i));
	#else
		for (int i = 0; i < drawList.length; i++) {
			ptr = ((disp*)touch(&drawList,i));
	#endif
		glTranslatef(ptr->xT,ptr->yT,ptr->zT);
//		glRotatef(1,ptr->xR,ptr->yR,ptr->zR);

		ptr->obj->draw();

//		glRotatef(1,-ptr->xR,-ptr->yR,-ptr->zR);
		glTranslatef(-ptr->xT,-ptr->yT,-ptr->zT);
	}
	drawObject();

}

void jCartesian::update() {
	jObject *ptr;
```

```
	#ifdef useCF
		for (int i = 0; i < CFArrayGetCount(drawList); i++) {
			ptr = (jObject*)((disp*)CFArrayGetValueAtIndex(drawList,i))->obj;
	#else
		for (int i = 0; i < drawList.length; i++) {
			ptr = (jObject*)((disp*)touch(&drawList,i))->obj;
	#endif
		ptr->update();
	}

	updateObject();

	return;
}

void *jCartesian::add(jObject *adde) {
	disp *test = (disp*)malloc(sizeof(disp));
	test->xT=0;
	test->yT=0;
	test->zT=0;
	test->xR=0;
	test->yR=0;
	test->zR=0;
	test->obj=adde;
	#ifdef useCF
		CFArrayAppendValue(drawList,test);
	#else
		push(&drawList, test);
	#endif
	return adde;
}
void *jCartesian::add(jObject *adde, float X, float Y, float Z) {
	disp *test = (disp*)malloc(sizeof(disp));
	test->xT=X;
	test->yT=Y;
	test->zT=Z;
	test->xR=0;
	test->yR=0;
	test->zR=0;
	test->obj=adde;
	#ifdef useCF
		CFArrayAppendValue(drawList,test);
	#else
		push(&drawList, test);
	#endif
	return adde;
```

```
}
void *jCartesian::add(jObject *adde, float X, float Y, float Z, float Rx, float Ry, float Rz) {
	disp *test = (disp*)malloc(sizeof(disp));
	test->xT=X;
	test->yT=Y;
	test->zT=Z;
	test->xR=Rx;
	test->yR=Ry;
	test->zR=Rz;
	test->obj=adde;
	#ifdef useCF
		CFArrayAppendValue(drawList,test);
	#else
		push(&drawList, test);
	#endif
	return adde;
}
```

jCaps.h

```
/*
 *  jCaps.h
 *  Module
 *
 *  Created by Jon Snelling on 2/4/06.
 *  Copyright 2006 JumpMast3r. All rights reserved.
 *
 */

//#if defined(__APPLE__) || defined(MACOSX)
//	#include <GLUT/glut.h>
//#else
//	#include <GL/glut.h>
//#endif
//
//#ifndef bool
//#define bool char
//#define true 1
//#define false 0
//#endif

//#define JGL_
#define JLG_IDLE   0b000010
```