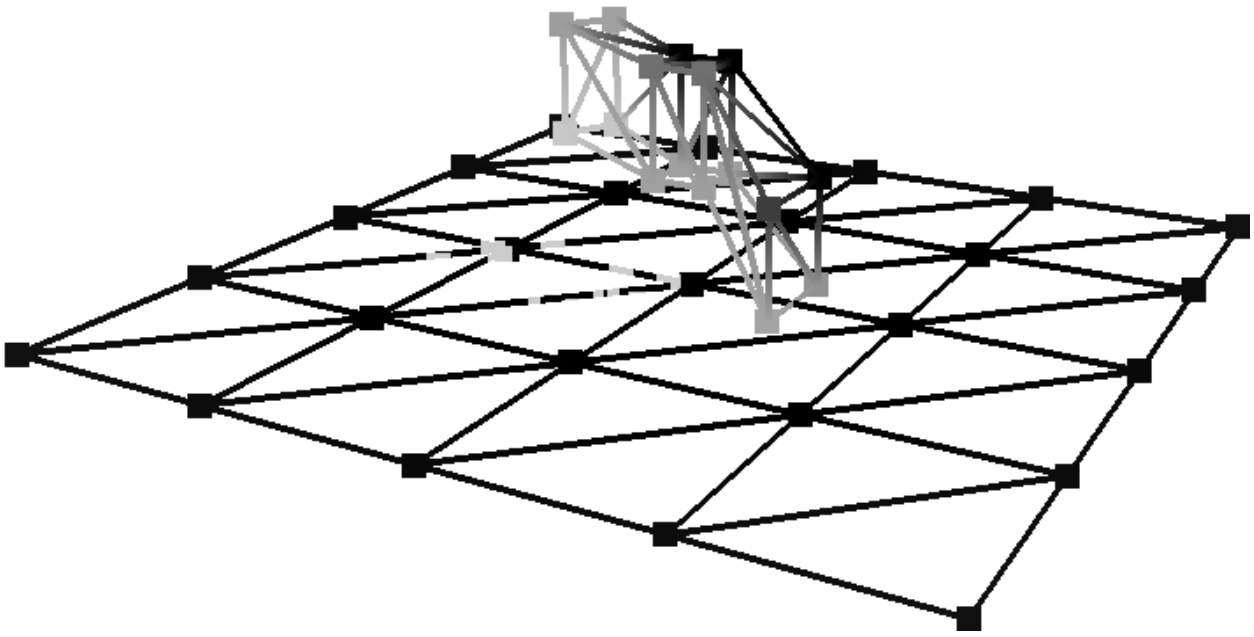
Plasma Works 3D Engine
Design Document

Draft 1 – 2/19/1900



Problem:

Plasma Works, is currently working on a game, named *Storm Saga*, which requires a three-dimensional object rendering engine (*3D engine*). The project is in its mid-design phase and Plasma Works has been planning to license a third party 3D engine. However, the company is interested in an analysis of the value and feasibility of the system and in having experimental spherical coordinate 3D technology developed into a rendering engine. To the knowledge of Plasma Works, a spherical coordinate system has not been implemented in the commercial world. A few examples of why people have not created a spherical 3D engine are:

- Mathematicians have defined matrices to deal with Cartesian coordinates for a while so all matrix transformations are done using Cartesian coordinates. Cartesian coordinates are not necessary, but in order to use spherical coordinates you would have to rewrite the matrices.
- Cartesian coordinates are more intuitive and easy to use. People are used to thinking in x,y,z locations in space, rather then distance and angle for the origin.
- Translations are difficult with spherical coordinates. It involves lots of trigonometry to translate a spherical coordinate compared with the simple addition of the Cartesian coordinates.

Proposed Solution:

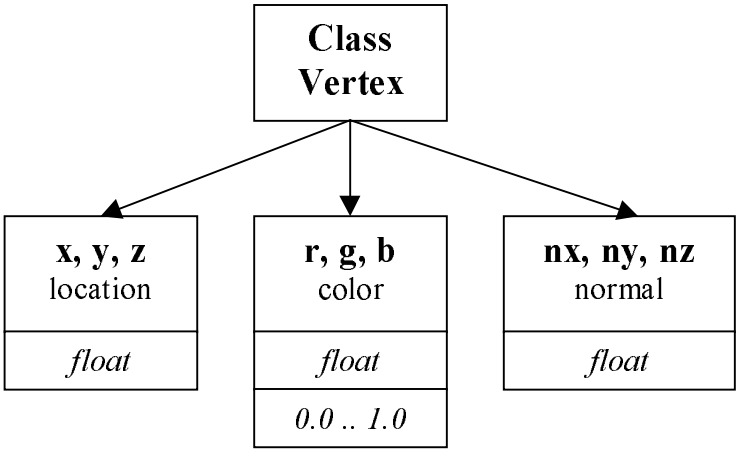
The *PlasmaWorks 3D Engine* will be developed for Abe Pralle that will serve as a test of the *Spherical Coordinate vs. Cartesian Coordinate* research ideas. It will contain all of the functionality Plasma Works has requested and be delivered on April 28, 2000. The 3D engine will have an understandable interface and use the latest hardware acceleration provided through the DirectX and OpenGL graphics libraries.

Capabilities:

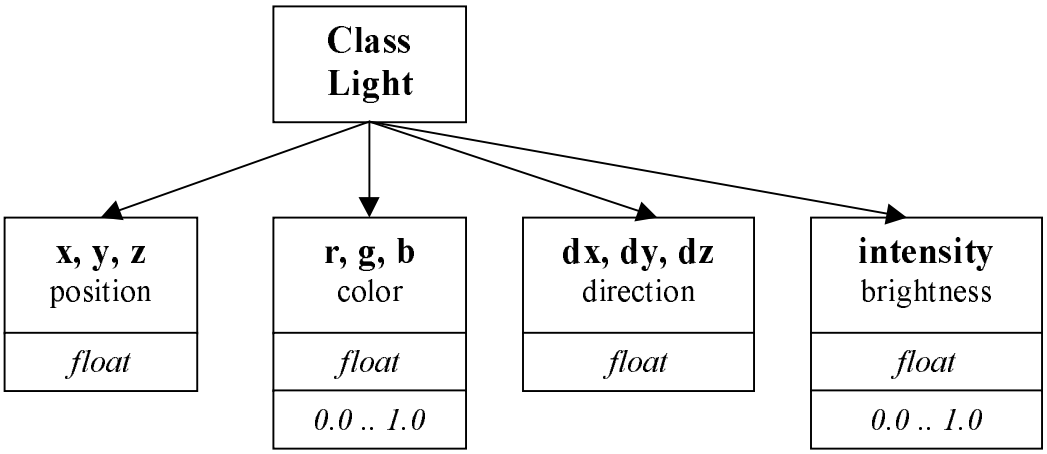
The following functionality will be available for the 3D Engine:

Item #	Requirement	Importance	Difficulty
1.	Compible under Windows 95/98	1	4
2.	Direct3d optimized	1	2
3.	Spot lighting	2	3
4.	Bulb lighting	2	3
5.	Directional (sunlight) lighting	2	3
6.	Standard texturing	2	3
7.	Wrapping texturing	2	3
8.	Mapped (tiled) Texturing	2	3
9.	Weighted vertices	3	2
10.	Mesh based model animation	2	2
11.	Keyfame based animations	2	2
12.	Model translation/rotation/scaling	2	5
13.	Generic DrawObjectAt(ScreenCoords) function	3	3
14.	Free floating camera	2	4
15.	Spherical coordinate system	2	3
16.	Cartesian coordinate system	2	4
17.	Object/polygon culling	2	4
18.	Lens flare	5	3
19.	Fire effects	5	3
20.	2D Bill-boarding	4	4
21.	Fogging	4	2
22.	Colored lighting	3	3
23.	Realistic shadows	4	1
24.	Fast shadows	2	2
25.	Panoramic sky texture	2	3
26.	Transparency	4	2
27.	Custom 3D modeler	3	1
28.	Texture editor/applier	3	2
29.	Demonstration movie exhibiting engine abilities	4	2

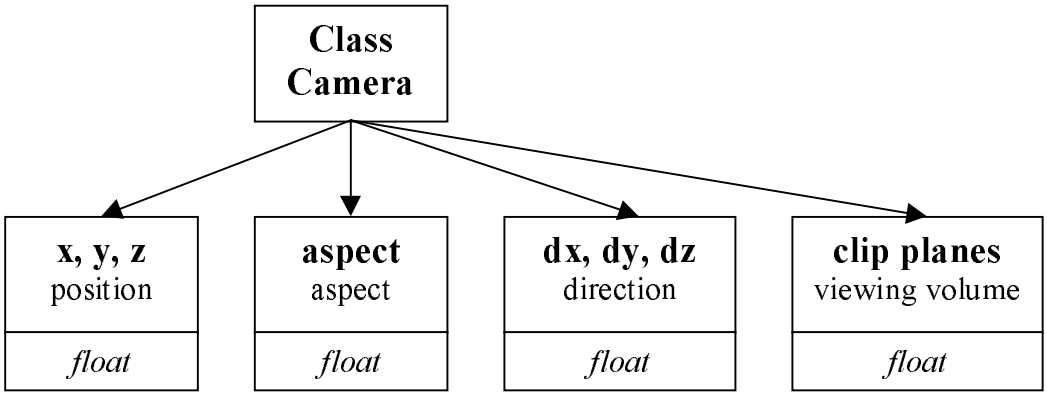
Architecture: Data View



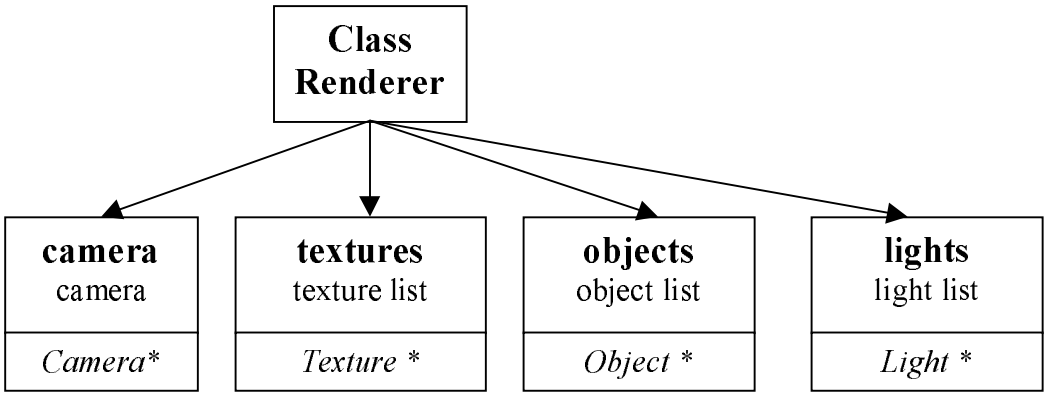
Vertices are made up of x, y, z coordinates, an nx, ny, nz normal vector, and a color.
Triangles are made up of vertex indices, 3 per triangle.
Texture Coords are made up of tu, tv values for each indice of each triangle.
Frames are made up of Vertex offset values (i.e. how far each vertex moves each frame).
Texture Number is a single integer value to identify what texture is being used for the object.



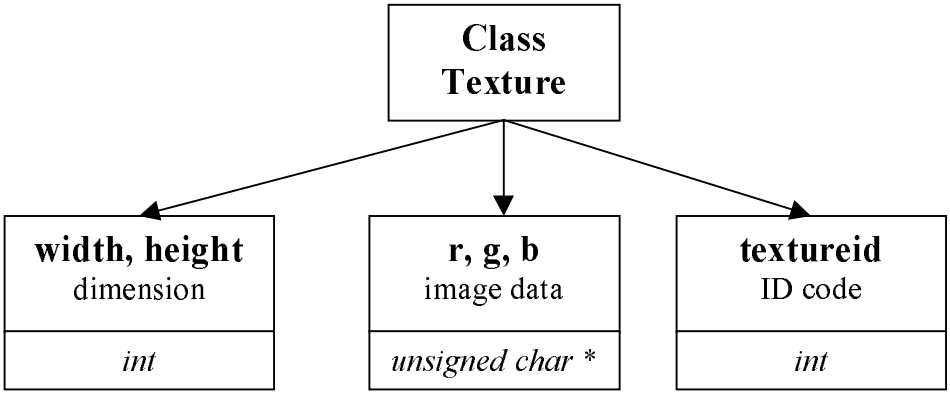
Position is made up of an x, y, z coordinate.
Direction is made up of an x, y, z vector.
Intensity is a single floating point from 0.0 to 1.0.
Color is made up of 3 r, g, b values from 0.0 to 1.0 each.



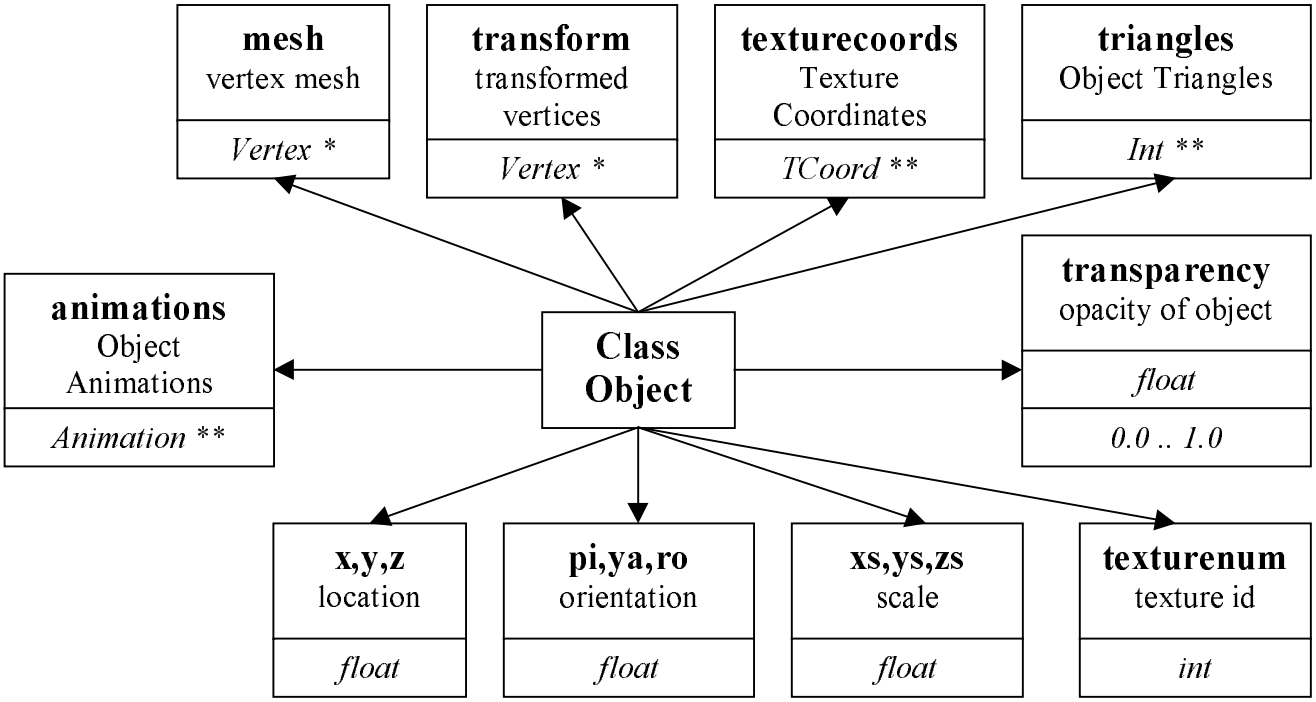
Position is made up of an x, y, z coordinate.
Direction is made up of an x, y, z vector.
Aspect tells how the camera lens works (w/ respect to the screen resolution).
Clip Planes are the front and rear clipping planes of the camera's viewing volume.



Camera is the camera the 3D renderer uses to render through.
Textures are the actual image data with corresponding ID codes.
Objects are the pointers to all the objects in the rendering pipeline.
Lights are the list of lights to be used to illuminate the scene.

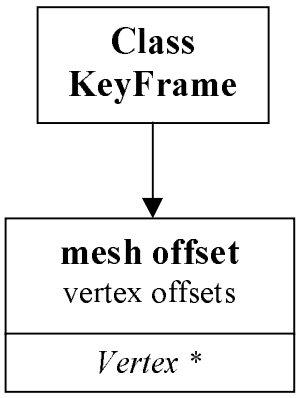


Color is the array of colors that make up the texture.
Dimension is the width and height of the texture.
Texture ID is the integer identifier used to refer to the texture by outside classes.

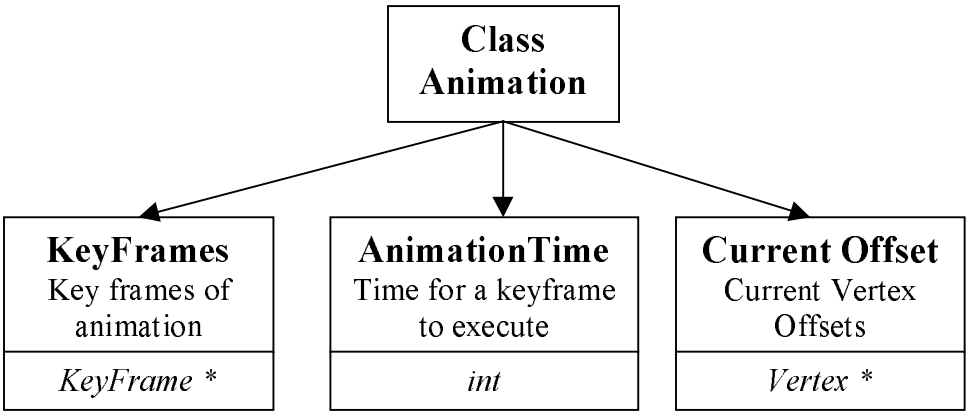


Mesh is the array of vertices that make up the object.
Transform is the array of vertices transformed to world-space.
Texture Coords is the array of texture coordinates for each triangle.
Triangles is the array of triangles that define the surface of the object.
Animations is the array of animations the object is capable of.
Transparency measures the opacity of the object.
X, Y, Z is the object's location in world-space.
Pi, Ya, Ro is the orientation of the object in world-space.

XS, YS, ZS is the object's scale in world space.
Texture Num is the texture that the object is using.

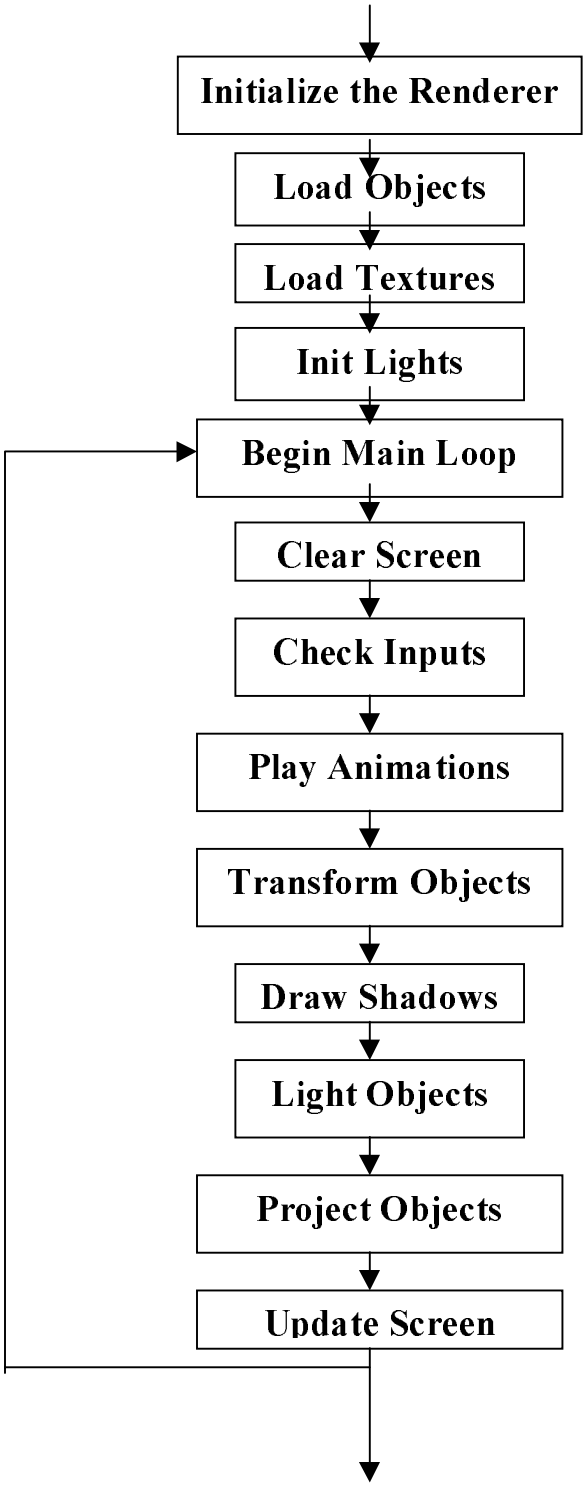


Mesh Offset is the distance each vertex must move for this frame of animation.

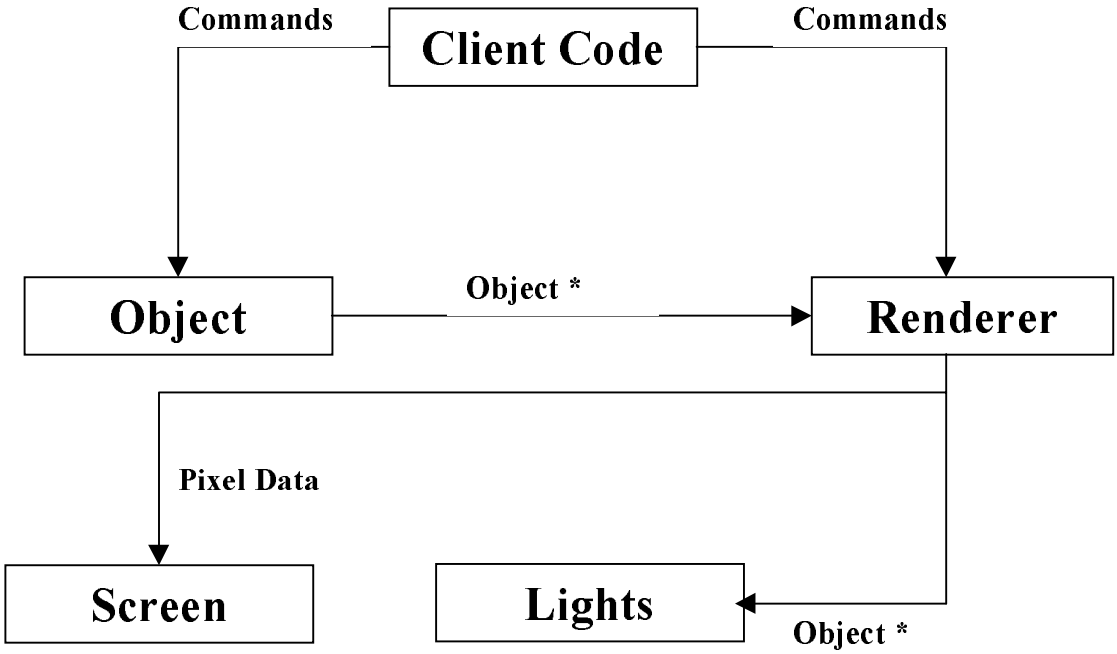


KeyFrames is the array of keyframes that make up the animation.
AnimationTime is the time it takes for one keyframe of the animation to be displayed.
Current Offset is the array of vertices representing the current keyframe offset being displayed.

Architecture: External Interface



Architecture: Data Flow Diagrams



Architecture: Class Task Lists

Vertex:

- Constructor with defaults initializes location to 0.0, normals to 0.0, and color to white
- Constructor with values for location then color
- Return locations, normals, and color values
- Set locations, normals, and color values

Light:

- Default constructor initializes all values to 0.0
- Light an object* vertices
- Set light color, intensity, position, or direction depending on light type.
- Return light color, intensity, position, or direction depending on light type.

Camera:

- Default constructor starts the camera at 0, 0, 1 pointing at 0, 0, 0
- Set the camera position, direction, and aspect (viewing frustum)
- Return the camera position, direction, and aspect
- Move the camera left, right, up, down, forward and backward.
- Pan the camera left, right, up and down
- Rotate the camera around a point

Renderer:

- Setup the display hardware depending on input parameters
- Set the screen resolution
- Add an object to the render pipeline
- Render an object in shadow mode(black with set transparency)
- Render all objects in the pipeline
- Render objects in line mode
- Render objects in point mode
- Render objects in textured mode
- Light objects in the render pipeline
- Disable/Enable transparency
- Disable/Enable anti-aliasing

Texture:

- Default constructor initializes all arrays to NULL
- Load texture from .BMP file
- Set Texture ID
- Get image information

Object:

- Move/Rotate/Scale the object in world-space
- Change object opacity

- Transform vertices to world-space
- Project vertices to a specified plane
- Play/Stop Animations
- Compute the normals of its vertices.
- Write object makeup to a file.

KeyFrame:

- Load in a collection of offset Vertices.
- Return the set of offset vertices

Animation:

- Load in Keyframes from a file.
- Play an animation
- Return the time for one frame or one animation.

Architecture: Class Descriptions

Vertex:

Vertices are the basic building blocks of the 3D Engine. They contain a coordinate location in 3-space, a normal vector for lighting, and a color for the lights to modify. All objects are made up of a collection of vertices that are connected to form solid surfaces.

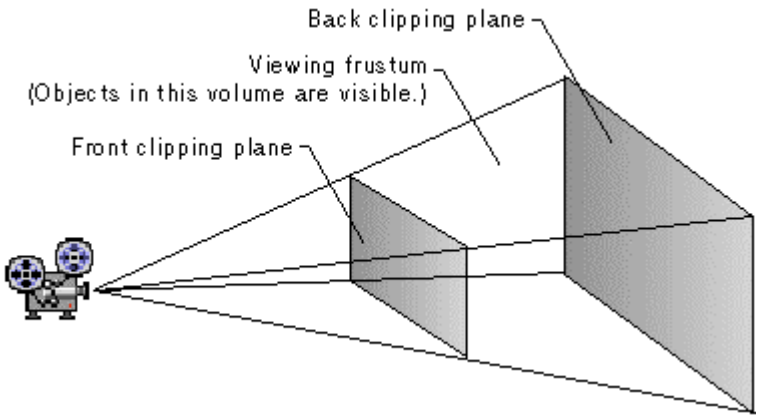
Light:

Lights help illuminate the 3D scenes created by the Client code. There are 4 types of lights; ambient, directional, spot, and bulb. All lights have a color and the ability to light up the vertices of an object depending on the object location/orientation and the light parameters.

Ambient lights light up the scene regardless of object location. Directional lights are like the sun, they have a direction, but are considered infinitely distant and therefore light all objects equally. Spot lights are lights with both a direction and a position. Bulb lights have only a position, they light equally in all directions.

Camera:

The camera determines where and how the 3D scene is viewed by the player. The camera can be moved around anywhere in the scene (it is "free floating") and pointed in any direction. Depending on the position and direction of the camera, the objects viewed by the player change. The camera also has an aspect. The aspect of the camera determines how the camera viewing area changes with respect to width and height. Finally, the camera has a viewing frustum. The viewing frustum has a front clipping plane and a back clipping plane. This makes up it's viewing volume. Any object not within the viewing volume is clipped.



Renderer:

The Renderer is the cat's pajamas. Two versions will be included with the 3D engine, a DirectX version and an OpenGL version. We will concentrate on the OpenGL version initially, and proceed to create the DirectX version as time permits. Both versions will be equally capable. The Renderer sets up the hardware behind the scenes to be ready to draw 3D objects. It does the

drawing of the 3D objects to the screen. It contains all of the lights and is responsible for passing pointers of the objects to the light classes for lighting.

The Renderer can draw objects in line, point, textured mode, or any combination of the three. It is capable of drawing objects in shadowed mode as well. This consists of drawing the objects as the color black, with no textures, and at a set level of transparency.

Texture:

The texture class holds a bitmap image that the Renderer will use to texture the objects in the pipeline. It is capable of reading .BMP files and loading in image data. It is primarily used for storing data.

Object:

The Object class is really where the fun begins. Objects are the things that actually get drawn to the screen, moved around, exploded, inflated, etc. They are the life of the 3D engine party. Objects can be moved around, animated, rotated, and generally manipulated as you would an object in reality.

KeyFrame:

A Keyframe is one point of an animation. A KeyFrame defines a set of vertex offsets telling how far each vertex should move from where they are to reach the current KeyFrame. Each time an animation is played, it looks at the last KeyFrame and generates all the in between points to the next KeyFrame.

Animation:

An animation is a sequence of KeyFrames that are all blended together into a smooth movement, thereby bringing the models to life. Animations contain a series of KeyFrames, and when the animation is played they generate the proper offsets from the first KeyFrame to the last.

Header Files: Vertex

```
/*****************************************************************************

  File                  :       vertex.h
  Version               :       .1
  Programmer            :       Craig Post
  Copyright             :       (c) Plasma Works
  Purpose               :       Definition for the VERTEX Class.
  Description           :       The vertex is the basic building block of the models
                                        in the 3D engine.

  Modification Record:
        2/19/1900       Original Release                                Craig Post

*****************************************************************************/
/*****************************************************************************
    Public Variables:

                float m_fX, m_fY, m_fZ;
                        - These variables contain the location in
                          3-space of the vertex in question.

                float m_fNX, m_fNY, m_fNZ;
                        - These variables define the normal vector coming off
                          of the vertex.  This is used in lighting.

                float m_fRed, m_fGreen, m_fBlue;
                        - The red, green, and blue values for color are used in
                          shading the vertex.

*****************************************************************************/
/*****************************************************************************
    Public Functions:

                Vertex();
                        - Default constructor.  All values are initialized to zero,
                          except color, which is initialized to 1.0.

                Vertex(float xl, float yl, float zl);
                        - This constructor initializes the location of the vertex
                          to xl, yl, zl.

                void SetLoc(float xl, float yl, float zl);
                        - This funciton sets the
                          location of the vertex to xl, yl, zl.

                void SetColor(float red, float gre, float blu);
                        - This funciton sets the color of the vertex to red, gre, and
                          blu for r,g,b respectively.

                void SetNormal(float xn, float yn, float zn);
                        - This function sets the normal vector nx,ny,nz to
                          xn,yn,zn respectively.

                float GetX();
                        - Returns the x location.

                float GetY();
                        - Returns the y location.

                float GetZ();
                        - Returns the z location.

*****************************************************************************/

#ifndef PLASMA_VERTEX__H
#define PLASMA_VERTEX__H

class Vertex
{
```

```
public:

        //location
        float m_fX, m_fY, m_fZ;
        //normal vector
        float m_fNX, m_fNY, m_fNZ;
        //color values
        float m_fRed, m_fGreen, m_fBlue;

        //default constructor
        Vertex();

        //constructor with location defined
        Vertex(float xl, float yl, float zl);

        //change the location of the vertex
        void SetLoc(float xl, float yl, float zl);

        //change the color of the vertex
        void SetColor(float red, float gre, float blu);

        //change the normal of the vertex
        void SetNormal(float xn, float yn, float zn);

        //return the x location
        float GetX();

        //return the y location
        float GetY();

        //return the z location
        float GetZ();

};

#endif
```

Header Files: Light

```
/*****************************************************************************

  File                  :       light.h
  Version               :       .1
  Programmer            :       Craig Post
  Copyright             :       (c) Plasma Works
  Purpose               :       Definition for the LIGHT Class.
  Description           :       The initial light class. All lights are derived from class
                                Light, which contains color and Intensity. The derived LIGHT
                                classes are: BULBLIGHT, DIRECTIONALLIGHT, SPOTLIGHT,
                                and AMBIENTLIGHT.

  Modification Record:
       2/19/1900       Original Release                          Craig Post

*****************************************************************************/

/*****************************************************************************

        PUBLIC MEMBER FUNCTIONS:

                for Class LIGHT:

                        Light(int r, int g, int b);
                                - The base light constructor takes in the color of the light as
                                parameters.

                        Light();
                                - The default constructor initializes the light to a white
                                  light (1.0, 1.0, 1.0).

                        void SetColors(int r, int g, int b);
                                - Set the color of the light to r, g, and b.

                        int GetRed();
                                - Return the red part of the light color.

                        int GetGreen();
                                - Return the green part of the light color.

                        int GetBlue();
                                - Return the blue part of the light color.

                for Class BULBLIGHT:

                        BulbLight(int r, int g, int b);
                                - The bulb light takes in a color and an intensity as parameters.

                        BulbLight(int r, int g, int b, float xl, float yl, float zl);
                                - This constructor takes the color of the bulb, the position of the
                                bulb and the intensity.

                        BulbLight(float xl, float yl, float zl);
                                - This constructor takes the position and intensity of the
                        light as parameters.  Color is initialized to white.

                        BulbLight();
                        - The default light appears at 0,0,0 with color of white and
                          intensity of 1.

                        void MoveLightBy(float xoff, float yoff, float zoff);
                                - Move the bulb by a position offset.

                        void MoveLightTo(float xoff, float yoff, float zoff)
                                - Move the bulb to a position in world space.

                for Class DIRECTIONALLIGHT:
```

```
                DirectionalLight(int r, int g, int b);
                        - This constructor takes the color as its parameters.

                DirectionalLight(int r, int g, int b, float
                                xd, float yd, float zd);
                        - This constructor takes in the color and the direction
                          of the directional light.

                DirectionalLight(float xd, float yd, float zd);
                        - This constructor takes in the direction of the light and
                          initializes the color to white.

                DirectionalLight();
                        - The default constructor points the light in the direction
                          0,0,0 with a color of white.

                void Normalize();
                        - Normalizes the direction (making it a unit vector).

                void SetDirection(float xd, float yd, float zd)
                        - Sets the direction of the light to x,y,z.

        for Class AMBIENTLIGHT:

                AmbientLight(int r, int g, int b);
                        - Initialize the light to color r,g,b.

                AmbientLight();
                        - Initialize the light to color white.

        for Class SPOTLIGHT:

                SpotLight(int r, int g, int b);
                        - Initialize the light to the color passed in and direction
                          and position are initialized to 0,0,0 each.

                SpotLight(int r, int g, int b, float x, float y, float z,
                                float xd, float yd, float zd);
                        - This constructor initializes all variables to the passed
                          in parameters.

                SpotLight(float x, float y, float z, float xd, float yd, float zd);
                        - The color is initialized to white, while the other varaibles are
                        initialized to the passed in a parameters.

                SpotLight();
                        - The default constructor initializes the color to white
                          and the other variables to 0.

                void Normalize();
                        - Turn the direction of the light into a unit vector(of length 1).

                void SetDirection(float xd, float yd, float zd);
                        - Set the direction of the light.

                void SetPosition(float x, float y, float z);
                        - Set the position of the light.

PUBLIC MEMBER VARIABLES:

        for Class LIGHT:
                enum LightType;
                        -  defines four different types of light.
                int m_fRed, m_fGreen, m_fBlue;
                        - holds a float value 0.0 to 1.0.
                float m_fIntensity;
                        - defines light brightness as 0.0 to 1.0


        for Class BULBLIGHT:
                float m_fX, m_fY, m_fZ;
```

```
                        -  location of the light source

                for Class DIRECTIONALLIGHT:
                        float m_fXD, m_fYD, m_fZD;
                                - direction of the light source

                for Class SPOTLIGHT:
                        float m_fX, m_fY, m_fZ;
                                - location of the light source
                        float m_fXD, m_fYD, m_fZD;
                                - direction of the light source

                for Class AMBIENTLIGHT:
                        none

*****************************************************************************/

#ifndef PLASMA_LIGHT__H
#define PLASMA_LIGHT__H

#include <math.h>

//This enumerated type holds the different types of lights.
enum LightType { bulb, dir, spot, amb };

class Light
{
public:

        //The color of the light.
        float m_fRed, m_fGreen, m_fBlue;

        //The brightness of the light.
        float m_fIntensity;

        //The type of light this one is.
        LightType m_ltType;

        //Constructors
        Light();
        Light(int r, int g, int b);

        //Return the individual color components of the light.
        int GetRed();
        int GetGreen();
        int GetBlue();

        //Set the light color.
        void SetColors(int r, int g, int b);

};

class BulbLight : public Light
{
public:
        //The bulb's position
        float m_fX, m_fY, m_fZ;

        //Constructors
        BulbLight(int r, int g, int b, float intensity)
        BulbLight(int r, int g, int b, float xl, float yl, float zl, float intensity)
        BulbLight(float xl, float yl, float zl, float intensity)
        BulbLight()

        //Action functions
        void MoveLightBy(float xoff, float yoff, float zoff)
        void MoveLightTo(float xoff, float yoff, float zoff)

};

class DirectionalLight : public Light
```

```
{
public:

        //The light's direction.
        float m_fXD, m_fYD, m_fZD;

        //Constructors
        DirectionalLight(int r, int g, int b);
        DirectionalLight(int r, int g, int b, float xd, float yd, float zd);
        DirectionalLight(float xd, float yd, float zd);
        DirectionalLight();

        //Turn the direction into a unit vector.
        void Normalize();

        //Set the direction of the light.
        void SetDirection(float xd, float yd, float zd);
};

class AmbientLight : public Light
{
public:

        //Constructors
        AmbientLight(int r, int g, int b);
        AmbientLight();
};


class SpotLight: public Light
{
public:

        //The position of the spot light.
        float m_fX, m_fY, m_fZ;

        //The direction of the spot light.
        float m_fXD, m_fYD, m_fZD;

        //Constructors.
        SpotLight(int r, int g, int b);
        SpotLight(int r, int g, int b, float x, float y, float z,
                          float xd, float yd, float zd);
        SpotLight(float x, float y, float z, float xd, float yd, float zd);
        SpotLight();

        //Turn the lights direction into a unit vector.
        void Normalize();

        //Set the direction of the light.
        void SetDirection(float xd, float yd, float zd);

        //Set the position of the light.
        void SetDirection(float x, float y, float z);
};

#endif
```

Header Files: Camera

```
/****************************************************************************

  File                  :       camera.h
  Version               :       .1
  Programmer            :       Craig Post
  Copyright             :       (c) Plasma Works
  Purpose               :       Definition for the CAMERA Class.
  Description           :       A camera object contains the camera that views the
                                3D scene set up by the client code.  It can be moved
                                around the 3D world like jelly through rain.

  Modification Record:
        2/19/1900       Original Release                                Craig Post

****************************************************************************/

/****************************************************************************

        PUBLIC MEMBER FUNCTIONS:

                Camera();
                        - The default camera is constructed pointing at the
                          origin and positioned at 5 back on the z axis.

                Camera(float xl, float yl, float zl,
                           float pitch, float yaw, float roll);
                        - Initialize the camera to a location and an orientation.

                void MoveBy(float xoff, float yoff, float zoff);
                        - Move the camera by an offset.

                void MoveTo(float xoff, float yoff, float zoff);
                        - Move the camera to a location.

                void RotateBy(float pitch, float yaw, float roll);
                        - Rotate they camera by an offset.

                void RotateTo(float pitch, float yaw, float roll);
                        - Rotate the camera by an offset.

                void MoveForward(float dist);
                        - Move the camera forward by dist.

                void MoveBackward(float dist);
                        - Move the camera forward by dist.

                void MoveRight(float dist);
                        - Move the camera right by dist.

                void MoveLeft(float dist);
                        - Move the camera left by dist.

                void MoveUp(float dist);
                        - Move the camera up by dist.

                void MoveDown(float dist);
                        - Move the camera down by dist.

                float GetX();
                        - Return the X position of the camera.

                float GetY();
                        - Return the Y position of the camera.

                float GetZ();
                        - Return the Z position of the camera.

                float GetPitch();
                        - Return the Pitch of the camera's rotation.
```

```
                float GetYaw();
                        - Return the Yaw of the camera's rotation.

                float GetRoll();
                        - Return the Roll of the camera's rotation.

        PUBLIC MEMBER VARIABLES:
                none.

*****************************************************************************

        PRIVATE MEMBER FUNCTIONS:
                none.

        PRIVATE MEMBER VARIABLES:
                float m_fX, m_fY, m_fZ;
                        - The location of the camera.

                float m_fPitch, m_fYaw, m_fRoll;
                        - The orientation of the camera.

                float m_fAspect;
                        - The aspect of the viewport.  This is the ratio of the
                          viewport width to the viewport height.

                float m_fFrontClippingPlane;
                        - The front clipping plane.  Any objects before this
                          plane the camera will remove.

                float m_fRearClippingPlane;
                        - The rear clipping plane.  Any objects beyond this plane
                          will be clipped by the plane.

*****************************************************************************/
#ifndef PLASMA_CAMERA__H
#define PLASMA_CAMERA__H

class Camera
{
private:
        //position and orientation.
        float m_fX, m_fY, m_fZ;
        float m_fPitch, m_fYaw, m_fRoll;

        //Wierd camera shit.
        float m_fAspect;
        float m_fFrontClippingPlane;
        float m_fRearClippingPlane;

public:
        //Constructors.
        Camera();
        Camera(float xl, float yl, float zl,
                        float pitch, float yaw, float roll);

        //Camera translation/rotation functions.
        void MoveBy(float xoff, float yoff, float zoff);
        void MoveTo(float xoff, float yoff, float zoff);
        void RotateBy(float pitch, float yaw, float roll);
        void RotateTo(float pitch, float yaw, float roll);
        void MoveForward(float dist);
        void MoveBackward(float dist);
        void MoveRight(float dist);
        void MoveLeft(float dist);
        void MoveDown(float dist);
        void MoveUp(float dist);

        //Get functions.
        float GetX();
        float GetY();
```

```
        float GetZ();
        float GetPitch();
        float GetYaw();
        float GetRoll();

};

#endif
```

Header Files: Renderer

```
/*****************************************************************************

  File                :       renderer.h
  Version             :       .1
  Programmer          :       Craig Post
  Copyright           :       (c) Plasma Works
  Purpose             :       Definition for the RENDERER Class.
  Description         :       This file defines the Renderer class which is used for
                              all of the direct hardware stuff. Right now it's only
                              GameKit. Soon it should be OpenGL able

  Modification Record:
        2/19/1900               Original Release                    Craig Post

*****************************************************************************/

/*****************************************************************************

        FUNCTIONS GLOBAL TO THE CLASS:


                int APIENTRY WinMain(HINSTANCE hCurrentInst,
                                     HINSTANCE hPreviousInst,
                                     LPSTR lpszCmdLine,
                                     int nCmdShow);
                        -  enables windows-specific function calls by the operating system

                LONG WINAPI WindowProc(HWND hWnd, UINT uMsg,
                                       WPARAM wParam, LPARAM lParam);

                        -  enables windows to build a program-definable window

                HWND CreateOpenGLWindow(char* title,
                                        int x, int y,
                                        int width, int height);
                        -  enables windows to build a window in which OpenGL functions
                           can be used to draw.

                void Resize(int w, int h);
                          - windows function call to change the window size and redraw
                          the window

                int pw3DMain();
                        - called by 3D engine and used by the client to continually
                          update drawing space details.  this is where the main game
                          code will go.


        PUBLIC MEMBER FUNCTIONS:

                Renderer();
                        - default constructor

                Renderer(int sw, int sh);
                        - constructor which accepts desired "screen width" and "screen height" as
                        parameters

                void InitRenderer(int sw, int sh);
                        - re-initializes the renderer with new screen height and screen width
                                parameters

                void SetDrawPoints(bool dpoints);
                        - if dpoints is true, draw the vertices of the object

                void SetDrawTexture(bool dtext);
                        - if dtext is true, draw the textures which are mapped to the
                         surfaces of the object

                void SetDrawLines(bool dlines);
```

```
        - if dlines is true, draw the "outline" of the triangles which make
         up the object

void AddObjectToPipeline(Object * obj, int texturenum);
        - adds the object to the queue of objects to be drawn

void Render();
        - draws everything in the pipeline to the screen


void RenderAsShadows(float red, float green, float blue,
                float transparency);
        - draws eveything in the pipeline as a "shadow"; ie. at a constant
          color and a constant transparency

void ClearScreen();
        - clears the drawing area on the screen.

void ScreenSwap();
        - draw what is loaded into the video buffer to the screen;
          (decreases flicker)

void GetTexture(char * bmpname, int width, int height, int texturenum);
        - loads a texture into video memory for faster accessing

void PrintText(char * text, int x, int y);
        - draws a character string at a desired x and y location.

void SetHDC(HDC hdc)
        - inline function which assigns a global handle to the device context


void SetHWND(HWND hwnd)
        - inline function which assigns a global handle to the window

void CheckEvents();
        - updates the message queue for windows; bascially looks for
          keyboard input

bool Alive()
        - inline function which looks to see if the window has been killed

void PanCameraTo(float radians);
        - aim camera at a new point; ie. immediately changes the viewing
          area on the screen;

void PanCameraBy(float radians);
        - move the camera to a new point; slowly change the viewing area to
        the new poisition by moving the camera by an offset and displaying
        the new view; lokks like a film camera "panning" across a view from
        one point to another

void PointCameraAt(float x, float y, float z);
        - immediately point the camera at a new point and display the new view

void MoveCameraBy(float x, float y, float z);
                - slowly move the camera to the new position by increments

void MoveCameraTo(float x, float y, float z);
                - move the camera to a new position

void MoveCameraForward(float dist);
                - move the camera in the direction it's pointing by dist.

void MoveCameraBackward(float dist);
                - move the camera away from the direction it's pointing by dist.

void MoveCameraLeft(float dist);
                - sidestep the camera to the left by dist.

void MoveCameraRight(float dist);
```

```
                        - sidestep the camera to the right by dist.

                void MoveCameraUp(float dist);
                        - sidestep the camera up by dist.

                void MoveCameraDown(float dist);
                        - sidestep the camera down by dist.


        PRIVATE MEMBER FUNCTIONS:

                void DrawObject(Object * obj, int texturenum);
                        - draws the desired object in the window and adds the desired
                        texture to it.

                void DrawObjectAsShadow(Object * obj, float red, float green, float blue,
                                                        float transparency);
                        - draws an object as a uniform color with a uniform
                          transparency level


*****************************************************************************/
#ifndef PLASMA_RENDERER__H
#define PLASMA_RENDERER__H


// Windows includes
#include <windows.h>
#include <wingdi.h>
#include <winuser.h>

#include <stdlib.h>
#include <stdio.h>

// OpenGL includes
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glaux.h>

int APIENTRY WinMain(HINSTANCE hCurrentInst,
                                     HINSTANCE hPreviousInst,
                                     LPSTR lpszCmdLine,
                                     int nCmdShow);

LONG WINAPI WindowProc(HWND hWnd, UINT uMsg,
                                       WPARAM wParam, LPARAM lParam);

HWND CreateOpenGLWindow(char* title,
                                            int x, int y,
                                            int width, int height);

void Resize(int w, int h);
int pw3DMain();

// disable useless conversion warnings
#pragma warning (disable:4244)
#pragma warning (disable:4305)

#include "Object.h"

#define MAXOBJECTS 500
#define MAXTRIANGLES 10000


class Renderer
{
private:

        // screen dimensions
        int                     m_nScreenWidth;
        int                     m_nScreenHeight;
```

```
        // "what to draw" flags
        bool            m_bDrawPoints;
        bool            m_bDrawTextured;
        bool            m_bDrawLines;

    // pointers to objects and textures
        Object **       m_opObjectList;
        int *           m_npTextureList;

        // total number of drawable ojects
        int                     m_nCurrentNumObjects;

        // windows handles
        HDC                     m_hDC;
        HWND            m_hWnd;

        // windows "application alive" flag
        bool            m_bKillApp;

        // camera stuff
        float           m_fCamerapanangle;
        float           m_fCamerapandist;

        float           m_fCamerax;
        float           m_fCameray;
        float           m_fCameraz;

        float           m_fCameraAtx;
        float           m_fCameraAty;
        float           m_fCameraAtz;

        float           m_fCameraUpx;
        float           m_fCameraUpy;
        float           m_fCameraUpz;

        // private drawing functions
        void DrawObject(Object * obj, int texturenum);
        void DrawObjectAsShadow(Object * obj, float red, float green, float blue, float
transparency);


public:

        // drawing functions
        Renderer();
        Renderer(int sw, int sh);
        void InitRenderer(int sw, int sh);
        void SetDrawPoints(bool dpoints);
        void SetDrawTexture(bool dtext);
        void SetDrawLines(bool dlines);
        void AddObjectToPipeline(Object * obj, int texturenum);
        void Render();
        void RenderAsShadows(float red, float green, float blue, float transparency);
        void ClearViewport();
        void ScreenSwap();
        void GetTexture(char * bmpname, int width, int height, int texturenum);
        void PrintText(char * text, int x, int y);

        // functions to return handles
        void SetHDC(HDC hdc) { hDC = hdc; }
        void SetHWND(HWND hwnd) { hWnd = hwnd; }

        // check for keyboard input
        void CheckEvents();

        // check window "alive" status
        bool Alive() { return !KillApp; }

        // public camera motion functions
        void PanCameraTo(float radians);
```

```
        void PanCameraBy(float radians);
        void PointCameraAt(float x, float y, float z);
        void MoveCameraBy(float x, float y, float z);
        void MoveCameraTo(float x, float y, float z);
        void MoveCameraForward(float dist);
        void MoveCameraBackward(float dist);
        void MoveCameraLeft(float dist);
        void MoveCameraRight(float dist);
        void MoveCameraUp(float dist);
        void MoveCameraDown(float dist);
};

extern Renderer render;

#endif
```

Header Files: Texture

```
/*****************************************************************************

  File                  :       texture.h
  Version               :       .1
  Programmer            :       Craig Post
  Copyright             :       (c) Plasma Works
  Purpose               :       Definition for the TEXTURE Class.
  Description           :       Provides a vehicle for importing and utilizing textures
                                with 3D objects.

  Modification Record:
       2/19/1900               Original Release                     Craig Post

*****************************************************************************/

/*****************************************************************************

        PUBLIC MEMBER FUNCTIONS:

                Texture();
                        - The default constructor initializes the arrays and variables to
                          NULL.


                void LoadBitmapTexture(char *filename, bool ColorIsAvg, bool AlphaIsAvg,
                                                                        bool Reversed);
                        - Load a bitmap file into the texutre class.

                void SetupTexture();
                        - Setup the texture class.

                void ReSetupTexture();
                        - Re-load the texture.

        PUBLIC MEMBER VARIABLES:

                -- these three are constant values which are global to the class.
                        const int TF_NONE = 0;
                        const int TF_BILINEAR = 1;
                        const int TF_TRILINEAR = 2;

                unsigned char *m_ucpImageData;
                        - The actual chunk of pixel data, in bytes, of the texture.

                int m_nWidth;
                        - The width, in pixels, of the texture map.

                int m_nHeight;
                        - The height, in pixels, of the texture map.

                int m_nId;
                        - The unique identifier this texture uses.


*****************************************************************************

        PRIVATE MEMBER FUNCTIONS:
                none

        PRIVATE MEMBER VARIABLES:
                none

*****************************************************************************/
#ifndef PLASMA_TEXTURE__H
#define PLASMA_TEXTURE__H

#include "Renderer.h"
```

```
const int TF_NONE = 0;
const int TF_BILINEAR = 1;
const int TF_TRILINEAR = 2;

class Texture
{
public:

        //The raw image data of the texture.
        unsigned char *m_ucpImageData;

        //The texture dimensions in pixels.
        int m_nWidth;
        int m_nHeight;

        //the texture identifier.
        int m_nId;

        //Constructor
        Texture();

        //Loads a bitmap into the texture class.
        void LoadBitmapTexture(char *filename,
                                bool ColorIsAvg, bool AlphaIsAvg, bool Reversed);

        //set up the texture data for the class.
        void SetupTexture();

        //Re-initialize the texture data for this texture.
        void ReSetupTexture();

};

#endif
```

Header Files: Object

```
/*****************************************************************************

  File                :       object.h
  Version             :       .1
  Programmer          :       Craig Post
  Copyright           :       (c) Plasma Works
  Purpose             :       Definition for the OBJECT Class.
  Description         :       Version 1 of the object format brainstormed by Abe and
                              Craig.

  Modification Record:
       2/19/1900               Original Release                    Craig Post

*****************************************************************************/

/*****************************************************************************

        PUBLIC MEMBER FUNCTIONS:

                Object(int numverts,  int numtris, Vertex * m, int ** tri,
                               TCoord ** tcoords, int numanims, Animation ** anims);
                        - The object constructor takes a large series of parameters
                          in order to create an object.  The number of vertices, the
                          number of triangles, the array of vertices, the array of
                          triangles, the array of texture coordinates, number of
                          animiations, and the array of animations.
                ~Object();
                        - The destructor deallocates all of the dynamically
                          allocated arrays.

                void MoveBy(float xoff, float yoff, float zoff);
                        - Move the object by a certain offset.

                void RotateBy(float pitch, float yaw, float roll);
                        - Rotate the object by a certain offset.

                void MoveTo(float xoff, float yoff, float zoff);
                        - Move the object to a certain location.

                void RotateTo(float pitch, float yaw, float roll);
                        - Rotate the object to a certain rotation.

                void ScaleTo(float xsoff, float ysoff, float zsoff);
                        - Scale the object to a certain scale.

                void ScaleBy(float xsoff, float ysoff, float zsoff);
                        - Scale the object by a certain ammount.

                int ** GetTriangles();
                    - Returns a pointer to the the triangle arrray.

                TCoord ** GetTextureCoords();
                        - Returns a pointer to the texture coordinates for all of
                          the triangles.

                Vertex * GetTransVerts()
                        - Returns a pointer to the transformed vertices

                Vertex * GetMesh();
                        - Returns a pointer to the untransformed object mesh.

                int GetNumTriangles();
                        - Returns the number of triangles that make up the object.

                int GetNumVertices();
                        - Returns the number of vertices making up the objects.

                void ComputeNormals();
                        - Compute the normal vectors for each vertex in the mesh.
```

```
            void Transform();
                    - Transfrom all of the vertices out of local space and into
                      world space.  (Note:  The Vertex * transform array is
                      updated when calling this function)

            void ProjectObject(Camera * cam);
                    - Project the object's vertices to the screen with respect
                    to the camera passed in.

            void WriteObject(char * pobfile);
                    - Write the object to a .POB format.

            float GetZ();
                    - Returns the objects Z coordinate.

            void SetTextureNum(int tnum);
                    - Set the texture the object will use in rendering.

            int GetTextureNum();
                    - Return the texture the object is using.

            void SetTransparency(float trans);
                    - Set the transparency of the object (between 0.0 and 1.0).

            float GetTransparency();
                    - Returns the transparency setting of the object.

            void ProjectToPlane(float a, float b, float c, float d,
                                                  float dx, float dy, float dz);
                    - Project the objects vertices to a plane defined by
                    ax + by + cz + d = 0.
                    This is used in shadowing.

            void PlayAnimation(int animnum);
                    - Start playing the animation "animnum"

            void StopAnimation();
                    - Stop the currently playing animation.

        PUBLIC MEMBER VARIABLES:
                none.

*****************************************************************************

        PRIVATE MEMBER FUNCTIONS:
                void CopyVertices();
                        - Reset the transform vertices to the mesh vertices.

                void RotateVertices();
                        - Rotate the vertices to their world-space orientation.

                void TranslateVertices();
                        - Move the vertices to their world-space location.

                void ScaleVertices();
                        - Scale the vertices to their world-space scale.

        PRIVATE MEMBER VARIABLES:
                Vertex * m_vpMesh;
                        - The mesh of vertices that make up the object.

                Vertex * m_vpTransform;
                        - The transformed vertices of the mesh.

                int ** m_nppTriangles;
                        - The array of triangles that make up the surface of the object.

                TCoord ** m_tcppTextureCoords;
                        - The array of texture coordinates that show how textures
                          are mapped onto the object.
```

```
                int m_nNumVertices;
                        - The number of vertices making up the object.

                int m_nNumTriangles;
                        - The number of triangles making up the object.

                float m_fX, m_fY, m_fZ;
                        - The x,y,z location of the object in world space.

                float m_fPitch, m_fYaw, m_fRoll;
                        - The orientation{pitch, yaw, roll} of the object in world space.

                float m_fXs, m_fYs, m_fZs;
                        - The scale of the object in world space.

                int m_nTextureNum;
                        - The texture number being used by the object.

                float m_fTransparency;
                        - The transparency level of the object.

                Animation ** m_appAnimations;
                        - The array of animations of the object.

                int m_nNumAnimations;
                        - The number of animations the object is capable of.

                int m_nCurrentAnimation;
                        - The current animation being played.

*****************************************************************************/
#ifndef PLASMA_OBJECT__H
#define PLASMA_OBJECT__H

#include "vertex.h"
#include "light.h"
#include "camera.h"
#include "animation.h"

class Object
{
private:

                //The mesh of vertices
                Vertex * m_vpMesh;

                //The transformed vertices
                Vertex * m_vpTransform;

                //The triangles making up the surface of the triangle.
                int ** m_nppTriangles;

                //The texture coordinates of the object.
                TCoord ** m_tcppTextureCoords;

                //The number of vertices making up the object.
                int m_nNumVertices;

                //The number of triangles making up the object.
                int m_nNumTriangles;

                //The object position in world-space.
                float m_fX, m_fY, m_fZ;

                //The object orientation in world-space.
                float m_fPitch, m_fYaw, m_fRoll;

                //The object scale in world-space.
                float m_fXs, m_fYs, m_fZs;
```

```
                //The texture the object is using.
                int m_nTextureNum;

                //The transparency of the object.
                float m_fTransparency;

                //The array of animations the object contains.
                Animation ** m_appAnimations;

                //The number of animations the object contains.
                int m_nNumAnimations;

                //The currently playing animation.
                int m_nCurrentAnimation;

                //Reset the transformed vertices to the mesh.
                void CopyVertices();

                //Rotate all of the vertices to world-space orientation.
                void RotateVertices();

                //Move all of the vertices to world-space position.
                void TranslateVertices();

                //Scale the vertices to the world-space scale.
                void ScaleVertices();

public:
                //the constructor takes in pre-created arrays for vertices,
                //triangles, texture coords, and animations.
                Object(int numverts,  int numtris, Vertex * m, int ** tri,
                               TCoord ** tcoords, int numanims, Animation ** anims);

                //Destruction!!! YEAH!!!
                ~Object();

                //Move the object by an offset.
                void MoveBy(float xoff, float yoff, float zoff);

                //Rotate the object by an offset.
                void RotateBy(float pitch, float yaw, float roll);

                //Move the object to a position.
                void MoveTo(float xoff, float yoff, float zoff);

                //Rotate the object to an orientation.
                void RotateTo(float pitch, float yaw, float roll);

                //Scale the object to a set scale.
                void ScaleTo(float xsoff, float ysoff, float zsoff);

                //Scale the object by a certain ammount.
                void ScaleBy(float xsoff, float ysoff, float zsoff);

                //Return the triangle array.
                int ** GetTriangles();

                //Return the texture coords array.
                TCoord ** GetTextureCoords();

                //Return the transformed vertices.
                Vertex * GetTransVerts();

                //Return the vertex mesh.
                Vertex * GetMesh();

                //Return the number of triangles.
                int GetNumTriangles();

                //Return the number of vertices.
                int GetNumVertices();
```

```
                //Compute the normals for each vertex in the mesh.
                void ComputeNormals();

                //Transform the vertices from local space to world space.
                void Transform();

                //Project the vertices to screen coords.
                void ProjectObject(Camera * cam);

                //Write the object to a .POB file.
                void WriteObject(char * pobfile);

                //Return the Z value.
                float GetZ();

                //Set the texture number.
                void SetTextureNum(int tnum);

                //Return the objects texture number.
                int GetTextureNum();

                //Set the transparency level of the object (0.0 to 1.0)
                void SetTransparency(float trans);

                //Return the object transparency level.
                float GetTransparency();

                //Project the object to a plane (for shadowing)
                void ProjectToPlane(float a, float b, float c, float d,
                                                    float dx, float dy, float dz);

                //Start playing the animation #animnum.
                void PlayAnimation(int animnum);

                //Stop playing the current animation.
                void StopAnimation();
};

#endif
```

Header Files: KeyFrame

```
/****************************************************************************

  File                  :       keyframe.h
  Version               :       .1
  Programmer            :       Craig Post
  Copyright             :       (c) Plasma Works
  Purpose               :       Definition for the FRAME Class.
  Description           :       A "frame" object contains information for one frame
                                of animation

  Modification Record:
        2/19/1900       Original Release                            Craig Post

****************************************************************************/

/****************************************************************************

        PUBLIC MEMBER FUNCTIONS:

                KeyFrame(int numoffverts, Vertex * smesh);
                        - constructor creates the animation frame using the indicated mesh

                ~KeyFrame();
                        - destructor

                int GetNumVertices()
                        - inline funtion which returns the number of vertices
                          in the mesh

                Vertex *GetMeshOffset()
                        - inline function which returns the array of vertex offsets
                          for the mesh

        PUBLIC MEMBER VARIABLES:
                none


****************************************************************************/
#ifndef PLASMA_FRAME__H
#define PLASMA_FRAME__H

#include "vertex.h"

class KeyFrame
{
private:

        // amount each vertex should move in this frame
        Vertex * m_vpMeshOffset;

        // number of vertices in the mesh
        int m_nNumVertices;

public:

        // constructor
        KeyFrame(int numsubverts, Vertex * smesh);

        // destructor
        ~KeyFrame();

        //  functions to retrieve vertex and mesh offset info
        int GetNumVertices() { return NumSubVertices; }
        Vertex *GetMeshOffest() { return submesh; }

};

#endif
```

Header Files: Animation

```
/*****************************************************************************

  File                  :       animation.h
  Version               :       .1
  Programmer            :       Craig Post
  Copyright             :       (c) Plasma Works
  Purpose               :       Definition for the ANIMATION Class.
  Description           :       An "animation" is a sequence of "keyframes"
                                with interpolatated drawings displayed between them.
                                (which are defined in the KEYFRAME class)

  Modification Record:
        2/19/1900       Original Release                            Craig Post

*****************************************************************************/

/*****************************************************************************

        PUBLIC MEMBER FUNCTIONS:
                Animation(int timeforanim, int timeforoneframe);
                        - constructor; creates an animation which runs at the
                          indicated speeds

                ~Animation();
                        - destructor

                void LoadKeyFramesFromFile(char * framefile);
                        - loads the requested key frame information into memory

                void PlayAnimation(Vertex * transform);
                        - displays the requested animation sequence


                int GetTimeForAnimation() { return TimeForAnimation; }
                        - inline function which returns the length of time to
                          run the animation

                int GetTimeForOneKeyFrame() { return TimeForOneKeyFrame; }
                        - inline function which returns the length of time to
                          display a single frame of the requested animation

        PUBLIC MEMBER VARIABLES:
                none


*****************************************************************************/
#ifndef PLASMA_ANIMATION__H
#define PLASMA_ANIMATION__H

#include "frame.h"
#include "vertex.h"
```

```
class Animation
{
private:

        // array of keyframes in the animation
        KeyFrame ** m_kfppKeyFrames;

        // number of frames in the animation
        int m_nNumKeyFrames;

        // length of time to run the animation
        int m_nTimeForAnimation;

        // length of time to display one frame of the animation
        int m_nTimeForOneKeyFrame;

        // number frames displayed since the last keyframe was drawn
        int m_nCurrentTime;

        // index of the "next" to be drawn; this is the keyframe that is
        // currently being interpolated "to" since the drawing of the
        // most recent keyframe
        int m_nCurrentKeyFrame;

        // current offset from the drawing of the last keyframe
        Vertex * m_pCurrentOffset;

        // total number of vertices in the above array
        int m_nNumVertices;

public:

        // constructor
        Animation(int timeforanim, int timeforoneframe);

        //destructor
        ~Animation();

        // loads the animation frames into memory
        void LoadKeyFramesFromFile(char * framefile);

        // displays the animation sequence
        void PlayAnimation(Vertex * transform);

        // returns the length of time to run the animation
        int GetTimeForAnimation() { return TimeForAnimation; }

        // returns the length of time to display a single animation frame
        int GetTimeForOneKeyFrame() { return TimeForOneKeyFrame; }
};

#endif
```