E-234 Final Project Proposal - *Anamatem* a flexible OpenGL Game Engine with example *Asteroids* game.

Russell Lowke, Dec 15th 2004

Project Members: 1. Russell Lowke

Description:

One of the observations of many "Galaxian" or "Asteroids" style reflex computer games is that they generally demand the same collection of animation requirements, and, if these requirements are all collectively bundled into a single gaming engine, it should be possible to rapidly develop a wide variety of games simply by modifying parameters sent to the engine. Once this animation engine is developed, reflex games could be quickly developed by sending parameters, mostly collections of 3d meshes, 2d .tga images, and .wav sound files, to the engine according to rules specified from a relatively light code shell that controls aspects of the desired game.

The game engine, which I have tentatively called "Anamatem" would essentially be a parent object that manipulates a specified number of child objects to visually update themselves according to animation cycles; x, y, and z velocities; lighting; texture maps; sounds; and collisions with other child objects.

To cater for different speeds of machines, especially low end machines, each child object records the exact time since it's last update, and then updates according to time passed, ensuring consistent game play and delivery. Exact parameters to be sent to the engine *for each child object* would be as follows:

All parameters specified here may be "gotten" or "set" as need be by the game shell.

- #label_name A name assigned to this object for identification.

- #cycle_list. A list of 2d .tga image graphics or 3d image meshes that

describe the appearance of the object. 3d meshes can also contain texturing data and surface properties. The list might have only one entry, in which case the object will always appear the same, or many, in which case the engine will cycle through the list at a given rate of frames per second. The list can also include .wav sounds. When a sound is reached the Anamatem engine plays the sound immediately, and proceeds to the next image in the cycle_list.

- #frames_per_second — Specifies the number of frames per second to animate through the #cycle_list.

- #cycle_place — Specifies the current place in the cycle_list.

- #cycle_type — Specifies the animation cycle type, can be set to; #wrap, the cycle "wraps" to the beginning once the end is reached; #reverse, the cycle "reverses" direction once the end is reached; #end, the object deactivates once the end of its cycle is reached; #reverse_end, the object reverses and then deactivates once it gets back to the beginning of the cycle_list; #inactive, deactivates the object immediately.

- #cycle_direction — Will be +'ve (+1), -'ve (-1), or 0. Specifies the direction the engine is cycling through the cycle_list.

- #facing — The angle the object is facing in degrees for x, y and z components. If x = 0, y = 0, and z = 0 then the object is facing directly downward into the z axis.

- #rotation — Specifies the x, y, and z angular velocities of this object, if any.

• #bounding_box	The boundary box of this object, comprised of 8 points.
• #display_area	Specifies the maximum x, y, and z boundary area for this object. If the object reaches this boundary it behaves according to its boundary_type setting.
• #boundary_type	The display area boundary can be set to; #reflect, the object bounces off the display area, reversing its velocity and acceleration; #wrap, the object wraps to the other side of the display area; #deactivate, the object becomes inactive; #block, the objects location is block so not to exceed the display area.
• #velocity	The velocity of the object in x, y, z components, specified as pixels per second.
• #acceleration	The acceleration of the object in x, y, z components, specified in pixels per second per second.
• #friction	Amount of friction object encounters when moving.
• #max_velocity	The maximum velocity for this object.
• #location	The location of this object in x, y, z coordinate space.
• #destination	Holds the destination point of the object. Objects may be given a destination point to move to in a certain time or at a certain velocity.
• #duration	Holds the time at which the object "expires." Objects may be given a specific duration that they persist for, after which they deactivate.

- #collision_detection_list — A list of all the other objects that this object has to worry about colliding with. When a collision occurs a message is sent to the extension game shells handle_collision() method [see below].

- #turn_to_face — can be true or false. If true the object will always rotate to face the direction it is moving in, the direction being specified by its velocity. 2d .tga frames may optionally specify a collection of 2d .tga frames, one for each directional facing, the first being at 0 degrees, and the last being 0-(360/n) degrees.

- #smooth_turns — can be true or false. If true then the object animates smoothly through cycle_list, never jumping frames for any reason.

- #mimic — Sometimes it is desirable for an object to mimic the location of another, so that the two can appear as a single item with multiple moving parts. mimic specifies the object number, if any, to copy the location parameter from.

- #light_source — An object can hold parameters of a light source, shining a type of light in the direction the object is facing. All the light parameters are held in a light_source list.

The engine wall also have overall parameters such as:

- #camera_attached — The camera can be attached to an Anamatem object, getting its location and viewing direction from the objects location and facing. This parameter will hold the object number that the camera is attached to.

- #camera_look_at — The camera can be told to always look at a specific child object. That object is stored in this parameter.

- #update_frequency — The number of times per second that Anamatem will attempt to update all objects in its object list.

Aside from the ubiquitous get and set parameter methods, a few methods will be supplied to simplify the task of parameter setting, such as:

```
void package_cycle_list(vector<T> list);
void show_frames(int objectNum, CycleList list, float fps, Point location);
void go_to_point_by_time(int objectNum, Point destination, float duration);
void go_to_point_at_velocity(int objectNum, Point destination, float velocity);
void deactivateAll();
```

Also, every game shell *must* support the following four methods to handle events that are sent from the Anamatem engine.

- deactivated (int objectNum); — Tells the shell an object has deactivated.

- cycleFinished(int objectNum); — Tells the shell an object has exhausted its cycle_list. This will only happen if the cycle_type is set to #end.

- atDestination(int objectNum); — Tells the shell an object has reached its destination.

- handleCollision (int sourceObj, int targetObj, string behaveType); — Tells the shell that object sourceObj has collided with object targetObject and the label name for the type of collision is string behaveType

With such an engine in place it becomes a relatively simple task to construct a game such as Asteroids. In Asteroids each asteroid is assigned to an Anamatem child object as a mesh (crumpled sphere) with a random location, random velocity and random angular rotating velocity. The asteroids display_area is set to the bounds of the screen rectangle and the boundary_type set to #wrap. The players ship is assigned as a mesh that starts in the middle of the screen and accelerates toward the location of the mouse pointer. The ship's collision_detection_list is appended with each asteroid object. If a collision is detected with an asteroid the ships cycle_list is set to an explosion and to deactivate on the cycles end. When the ship fires a bullet a bullet object is created moving in the direction of the ships velocity. The bullet object is given a collision_detection_list of all the asteroids in play. If a bullet hits an asteroid the asteroid either deactivates, or spits into n smaller asteroids, each with random velocities and angular rotating velocity. Other elements can be introduced such as an alien ship that chases the player and shoots, or bonus items that add points if collected. Sounds can be embedded into the cycle lists or triggered by the game shell.

It is desirable to have a low end platform delivery requirement without accelerated video cards so potentially the widest possible audience can be reached. Also, I will be developing on low end machines without access to high end video cards--which is good for testing purposes. As such the 3d meshes will be low polygon count sprites for high performance and texture maps kept small and to a minimum. In some cases, such as the explosions, a cycle of 2d flat .tga bit maps will be used.

The hope is to later convert the program code to the Macintosh platform and to then be able to rapidly deliver high quality Windows and Macintosh arcade style games of a wide variety that are playable on a broad scope of computers, utilizing cross platform OpenGL, the C++ STL, and the above described Anamatem engine.

Russell Lowke