Animation System Design Document, v3

Randy Angle

October 19, 1999

Introduction

The animation system uses services provided by the sprites in the graphics system to build a new class called CAnimSprite. The bitmap information for the sprite changes at a regular animation rate to represent internal motion in a sprite object. For instance the arms and legs of a character. The animation format used will also incorporate delta frame compression, meaning each frame only stores the pixels that changed from the previous frame. The technique is similar to the popular FLI/FLC formats developed for Autodesk Animator Studio but extends the functionality to include WORD sized, instead of BYTE sized, pixels.

Each object can have several animations for different actions that the object can perform. The SAM (scene asset manager) will allocate assets and the object structure will point to the various action animations.

Requirements

Animation size is consistently one of the largest assets and memory requirements of any game. Animation size, both in memory and on disk is a significant concern. The optimal use of memory by compression is the way to counterbalance the size and achieve the required number of frames for the various animations in a typical scene.

The suggested frame rate for the PC target system is 30fps. The minimum system frame rate is recommended 15fps. On the MAC this minimum frame rate must also be achievable. The fact that each frame is difference compressed means that a significant amount of pixel writes is reduces as long as small changes to the image happen each frame. Having 20 animate-able objects on screen and 2 or 3 of them animating at anyone time should be doable with this system.

Structures/Classes

```
class CAnimSprite:CSprite
{
private:
        BYTE*   pCompData;      // location of the compressed frame data
        BYTE*   pFrameData;     // location of the next frame in the byte stream
        float   AnimTime;       // frame time roughly equivalent to 1/30 second so the
                                // granular size of a frame will remain that size
                                // so the frame displayed is AnimTime/30
        BOOL    bLooping;       // set TRUE if the animation loops
        BOOL    bDone;          // set when a non-looping animation finishes
public:
        CAnimSprite();
        ~CAnimSprite();
        void    Play(BYTE* pAnimData, BOOL bLoopingState);
        BOOL    GetLoopingState(void);
        BOOL    GetAnimDone(void);
        void    Update(float DeltaTime);
}
```

Schedule Task List

System Tasks	Duration	Dependent
Design CAnimSprite Class	1 Day	Design Document
Code CAnimSprite Class	4 Days	CAnimSprite Class designed
Integrate CAnimSprite Class	2 Days	CAnimSprite Class coded
Test & Revise CAnimSprite Class	1 Day	CAnimSprite Class integrated
Rework #1 CAnimSprite Class	1 Day	As Needed
Test & Revise CAnimSprite Rework #1	1 Day	CAnimSprite Class Reworked #1
Rework #2 CAnimSprite Class	1 Day	As Needed
Test & Revise CAnimSprite Rework #2	1 Day	CAnimSprite Class Reworked #2
Total	12 Days	

Memory

Besides the class size, which is relatively small, the CAnimSprite class uses a buffer that is loaded into system RAM for frame data. This should amount to less than 1K of RAM for class overhead and width*height*2/10 per frame. A typical 30 frame 64x64x2 animation might use approximately 24577 bytes of memory.

Risk Assessment

A real risk with the animation system is that it could be very slow or take too much RAM. A significant and necessary amount of time has been allocated to make this system a reliable and fast as possible. If necessary an alternative animation system based on some other form of compression might play faster on an MMX PC or PowerPC MAC, but there has been no R&D time allocated for that contingency.

QA & Test

For the animation system the QA department should pay special attention to the visual look of the on screen animating objects, the clipping behaviors at the edge of the screen, and the video frame rate on our minimum systems. One specific nag about animation is the looping condition. A looping animation should not jump or skip at the end of its' cycle. If this behavior is noticed, indicate it as a bug.