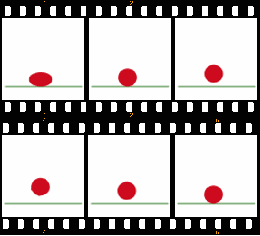
* Định dạng ảnh
* Định dạng MD2 – Animation
* Gắn âm thanh cho object
* Cấu trúc cho tower, tree
* Cấu trúc cho enemy, xử lí chuyển động cho enemy
* Tổ chức resource như model, sound
* Tổ chức quản lí các object trong in game
* Gắn AI

1.Animation

1.1 What is animation?

Arcording to wiki: “Animation is the rapid display of a sequence of images of 2-D or 3-D artwork or model positions in order to create an illusion of movement.”

Example: I have a set of images which present balls’ position following:



And when we display images rapidly (about 10 frames per second), we made illusion of ball’s movement.

1.2 How to use animation in WOT

In order to create the motion for the object in WOT like towers, enemies, we used a file whose type is MD2 (Quake 2 model). It stored all frames of model, frame by frame and with images texture at PCX or TGA type.

1.3 MD2 structure

1.3.1 Introduce

MD2 (Quake 2 Model) is a model format used by ID SOFTWARE’s id Tech 2 engine and is thus used by Quake II – first person shooter computer game, which is released on December 9, 1997 by Id Software.

The MD2 animations are achieved via key-frames on a per-vertex level. The key-frames are stored within model file and use interpolation to create smooth animation.

1.3.2 MD2 File Format

MD2 files are binary files which is store in little-endian. They begins with a fixed length header followed by static model data such as texture coordinates. Dynamic data such as vertices and normal are stored a number of file chunks call frames (or key-frames)

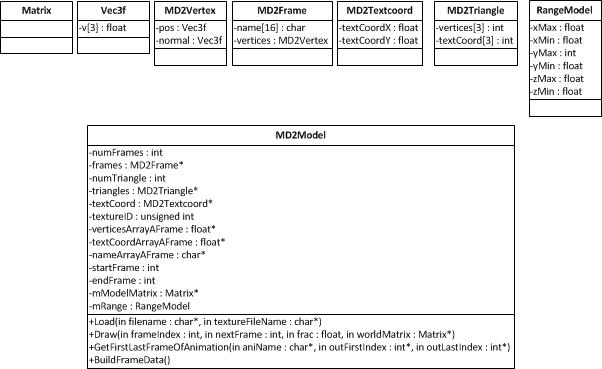
1.3.2.1 MD2 Header file

MD2 header file is fixed length part in md2 file, its size is 64bytes, included following information:

|  |  |  |  |
| --- | --- | --- | --- |
| Offset | Type | Name | Description |
| 0 | int | Ident | Magic number. Must be equal to "IDP2" |
| 4 | int | Version | MD2 version. Must be equal to 8 |
| 8 | int | SkinWidth | Width of the texture |
| 12 | int | SkinHeight | Height of the texture |
| 16 | int | FrameSize | Size of one frame in bytes |
| 20 | int | Num\_skins | Number of textures |
| 24 | int | Num\_vertices | Number of vertices |
| 28 | int | Num\_textcoord | Number of texture coordinates |
| 32 | int | Num\_triangle | Number of triangles |
| 36 | int | Num\_glcmds | Number of OpenGL commands |
| 40 | int | Offset\_skins | Offset to skin names (each skin name is an unsigned char[64] and are null terminated) |
| 44 | int | Offset\_vertices | Offset to frame data |
| 48 | int | Offset\_st | Offset to s-t texture coordinates |
| 52 | int | Offset\_tris | Offset to triangles |
| 56 | int | Offset\_frames | Vị trí nơi bắt đầu duyệt để lấy được tập frames. |
| 60 | int | Offset\_glcms | Offset to OpenGL commands |
| 64 | int | Offset\_end | Kết thúc header. |

Dự trên số lượng và vị trí các thông tin cần lấy từ file MD2 (vertices, textcoords, triangles, frames), ta sẽ thực hiện việc load và tổ chức model MD2 như sau.

1.3.2.2 Load và quản lí model MD2



Việc giao tiếp giữa định dạng file MD2 và chương trình sẽ được hiện thực thông qua lớp MD2Model. Cấu trúc lớp MD2Model và các struct hỗ trợ được thể hiện như hình trên. Sau đây tôi sẽ trình bày ý nghĩa các thông số và cách sử dụng lớp này.

Để tạo ra một model MD2 dùng cho các object của WOT ta cần hai thông tin là 2 đường dẫn đến 2 file, một là file md2 chứa thông tin của model, hai là file texture. Việc đọc được class MD2Model hiện thực và cung cấp thông qua hàm Load. Công việc của hàm load gồm những việc sau:

-Nhận diên file MD2 và đọc các thông tin từ md2 và load texture thông qua CImageManager.

-Thực hiện scale, translate nhằm đưa tất cả các model về vị trị (0,0) trong mặt phảng 0xy, có phương song song với Oz, và nằm trên phần dương của Oz. Kết quả sẽ được đưa vào ma trận mModelMatrix. Đồng thời tính toán mRange, nhằm xác định hình hộp chữ nhật bao quanh object để xử lí collision.

Ngoài ra, class MD2Model còn cung cấp thêm các hàm với các mục đích sau:

-Hàm GetFirstLastFrameOfAnimation với thông số đầu vào là aniName (tức là tên của action mong muốn model thực hiện như run, jump, attack, death,…) sẽ cho ra kết quả là frame đầu và frame cuối của action này (truyền con trỏ để giữ giá trị trả ra).

-Khi tính toán được frame đầu và frame cuối của 1 action, ta sẽ thông qua hàm Draw để render lên view.

- Bên cạnh đó, class MD2Model còn một một hàm khác nhằm giảm thiểu chi phí lúc render một model. Bởi vì, khi render, các đỉnh sẽ được tính và nội suy nhằm tạo một hiệu ứng di chuyển tốt nhất. Nhưng trong điều kiện phần cứng không cho phép, ta sẽ sử dụng hàm này để tính toán trước tất cả các vertice và lưu vào 3 mảng là: textCoordArrayAFrame, verticesArrayAFrame, nameArrayAFrame.

1.3.3 Quản lí các model

Việc quản lí các model MD2 được thực hiện nhờ một class Singleton là CModelManager. Đây là một class được thừa kế từ hai class là CSingleton và CResourceLookupTableI. Việc thừa kế CSingleton nhằm đảm bảo chỉ có đúng một đối tượng CModelManager tồn tại trong suốt quá trình game chạy. Còn đối với lớp CResourceLookupTableI để thừa kế lại cấu trúc dữ liệu này. Đây là một bản tra theo key, với key là integer.

Việc sử dụng CModelManager thông qua hai phương thức add và get với cú pháp như sau:

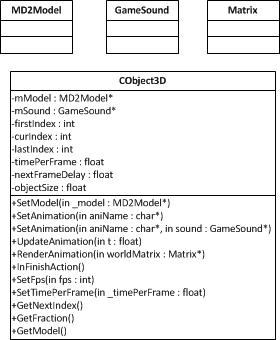
CModelManager::GetInstance()->(key, MD2FileName, textureFileName);

CModelManager::GetInstance()->Get(key);

2. Quản lí đối tương Game

2.1 CObject3D

Lớp CObject3D được tạo ra nhằm quản lí phần đồ họa và âm thanh của một object game. Được tổ chức như sau:



Các phương thức cơ bản để cài đặt một CObject3D là SetModel (xác định model sử dụng cho object này), SetAnimation (có 2 phương thức giúp người lập trình xác định việc có hay không dùng âm thanh cho object tương ứng với action vừa set).

Ngoài ra là các phương thức dùng để hỗ trợ cho việc render cho object.

2.2 Lớp GameObject

Đây sẽ là lớp lớp cha của tất cả các lớp object trong WOT. Mục đích của việc tạo ra lớp này nhằm đảm bảo sự thống nhất trong việc quản lí tất cả các object trong game.

Cấu trúc lớp này khá đơn giản chỉ gồm các thành phần như sau:



Trong đó, ObjectType là một enum nhằm xác định GameObject thuộc loại nào (enemy, tower, tree, bullet,..). Ngoài ra, lớp GameObject còn có 2 hàm virtual là Render() và Update(float tpf) nhằm đảm bảo việc con trỏ của lớp cha-GameObject có thể gọi tới các hàm Render và Update tương ứng của lớp con. Từ đó tiện cho việc quản lí của game.

2.3 Các lớp đối tượng Game

Hiện tại trong WOT, chúng tôi chỉ hiện thực vài lớp đối tượng cơ bản cho Object gồm: TowerObject, MapObject, TreeObject, EnemyObject, BulletObject. Mỗi lớp có mỗi điểm khác nhau nhằm đảm bảo việc thực hiện đúng chức năng của nó. Tuy nhiên chúng đều cùng thừa kế từ GameObject và hiện thực lại 2 phương thức ảo của lớp cha là Render và Update. Từ đó, việc quản lí các Object này trở nên dễ dàng hơn.

2.3.1 TreeObject

Đây là lớp đơn giản nhất trong các lớp Object của WOT. Cấu trúc như sau: