Peter Mileff PhD

Programming of Graphics

Introduction to 2D graphics

University of Miskolc
Department of Information Technology

MOVING OBJECTS…

Moving objects in 2D

- Better name for "moving texture": <u>**Object**</u>
  - More than just a texture
  - <u>Has several features:</u>
    - E.g. visible or not, movable, direction of rotation, etc.
  - The name is used preferentially in the game industry
    - Or some other equivalent
- <u>**Movement of an Object:**</u>
  - the shape (in this case an image) changes its position as a result of an event. E.g.: mouse movement, pressing a key
  - Position change has a direction vector and velocity that determine the nature of the movement

Moving objects in 2D

- **The theory of movement:**

  Object's new position(x,y) = current pos (x,y) + speed(v) * direction(x,y)

- **Continuous movement:**
  - In each frame, we perform the above operation for each object
  - Thereby movement will continuous
  - If the direction vector is a zero vector, the object stops.

Moving objects in 2D

The program main loop:

```
While ( !exit ) {
    HandleEvents();
    MoveObjects()
    DrawObjects();
}
```

The logic of the MoveObjects() function:

```
for (i =0; i < numOfObjects; i++) {
        Vector2 oldpos = obj[i].pos;
        obj[i].pos = oldpos + speed * direction;
 }
```

Moving objects in 2D

- **Advantage:**
  - The solution is very simple
- **Drawbacks:**
  - The solution is not efficient
- **The problem:** it occurs when we work with computers at very different speeds.
  - 1) If computer is slow, movement speed will be slow,
  - 2) If computer is fast, movement can be too fast
  - In case of early games it was typically observed phenomenon
    - E.g. in DOS age

ELAPSED TIME BASED MOVEMENT…

(TIME BASED MOVEMENT)

Time Based Movement

- Modified version(s) of the classic solution
- Ensures the same speed of moving objects
  - also on different speed machines
- **Background of the theory:**
  - Each graphic engine has a main loop (game loop) inside
  - This cycle runs faster on a fast computer and slower on a slow machine
  - **The Objective: measure the time between two main cycles**
    - we get a factor that can be used to standardize speed between machines
    - with a higher resolution timer (at least milliseconds)

Time Based Movement (Example)

```
while( game_is_running ) {
    prev_frame_tick = curr_frame_tick;
    curr_frame_tick = GetTickCount();
    elapsed_time = curr_frame_tick – prev_frame_tick;
    update( elapsed_time);
    render();
}
```

***GetTickCount()* function:**

returns milliseconds since the system was booted

Time Based Movement characteristics

- Your query is always OS dependent
- Ideally, a double precision floating-point number between 0 and 1
  - E.g.: 0.003568
- If the value is zero, then the timer resolution is not enough high
  - Cannot measure time between two frames
- **Zero value cannot be used!**
  - **Reason:** the factor will be included as a multiplication factor at the movements

obj[i].pos = oldpos + elapsed_time*(speed*direction);

Time Based Movement

- The multiplication factor affects the additive member of the position
- **<u>On a fast machine, this time is short:</u>**
  - so the additive tag will be smaller
  - Movement will be more continuous
- **<u>On slower machines this value is higher:</u>**
  - movement is less continuous
    - it may not be noticeable to the human eye
  - but the movement distance will be the same as the version running on the fast computer!

Time base movement

Extension 1

- **1. Maximizing the elapsed time:**
  - **Problem:** certain background processes in the operating system maybe use more resources
    - the elapsed time increases, resulting a larger “jump” in objects movement
  - **A typical example is debugging:** we stop the software for debugging,
    - restarting the software, the elapsed time will be very high if not maximized
  - **The objective: maximizing elapsed time**
    - for example to 1.0 value

Time base movement
Extension 2

- **2. “Smooting” the elapsed time:**
  - **The problem:** the elapsed time value may fluctuate between two graphically identical loop
    - Usually does not cause any problem in the software
  - However, it is advisable to compensate!
  - For example, calculate an average for the past and new loop:

```
elapsed_time += curr_frame_tick – prev_frame_tick;
    elapsed_time *= 0.5;
```

- Although the supplements are effective, they are not perfect.
- In some cases, it is also advisable to set a minimum or maximum FPS.

Animation in 2D…

Objektumok animációja

- Animation plays an important role in computer graphics
- This will make the software really "live"
  - E.g.: animation of a menu, window ot jumping shape
- **The classic animation:** to alternate a set of textures in a given sequence at a certain speed
- **Texture set:** is an array of textures that contains each phase of the animation
- In practice, an object consisting of textures is also called **Sprite**
- The more the phase, the more continuous the animation of the object will be when displayed

Example implementation

```
class CSprite {
        string mName                              // Sprite name
    vector<CSpriteFrame> mFrames;   // Frames vector
    int mNumFrames;                   // Number of frames
    int mActualFrame;                 // Actual frame
    Vector2 mPosition;                // position of the sprite
    Vector2 mScale;                   // Sprite scale value
    int mLastUpdate;                  // The last update time
    int mFps;                   // The number of frames per second
        float mZRotation;             // Z axis rotation value
public:
...
};
```

Example implementation

- **CSprite class:** a compact unit, which stores an animation sequence
- **It's components:**
  - **The name of the sprite:** important, because it is much easier to refer with a name
    - E.g.: getting the "jump" animation
  - **CSpriteFrame class:** stores a single frame
    - The SpriteFrame vector represents the animation
  - Position, size, rotation
  - Number of phases, current phase id
  - Animation speed

Example implementation

```
class CSpriteFrame {
    CTexture2D mFrame;                                  // Frame texture
    CString mName;                                      // Name of the frame
     vector<CBoundingBox2D> mBBoxOriginal;              // Original Bounding boxes
     vector<CBoundingBox2D> mBBoxTransformed;           // Transformed Bounding boxes

public:

    /// Default Constructor
    CSpriteFrame();
 …
};
```

Example implementation

- CSpriteFrame class:
  - Storing the images: **CTexture2D**
  - Name of the frame: sometimes can be useful
    - Referring by name is much easier!
  - Bounding box: for collision detection
    - original: it is important to keep it to speed up your calculations
    - transformed: the rotated, scaled and translated box of the original version

Example implementation

```
class CTexture2D {
    CVector2 mPosition;
    CVector2 mRotation;
    CVector2 mScale;
    bool bVisible;
    CVAOobject mTextureVAO;    // Storage data in VAO
    sColor mColor;             // Color information
    string mFilename;          // Holds the filename of the texture
    string mName;              // Name
    float mWidth;              //  Stores the width of the texture
    float mHeight;             // Stores the height of the texture
    unsigned int mTextureID;   // Holds the texture ID
    int mID;                   // Global ID of the texture
…
};
```

Example implementation

- <u>**CTexture2D class:**</u>
  - Position, rotation, translation, size
  - Filename
  - Name of the texture
  - Color information
  - <u>IDs:</u>
    - OpenGL ID: unique texture ID from the OpenGL
    - Global ID in the engine
  - Store vertex and texture coordinate in VAO

Store animation on the filesystem

- There are several ways to store animation images in the file system

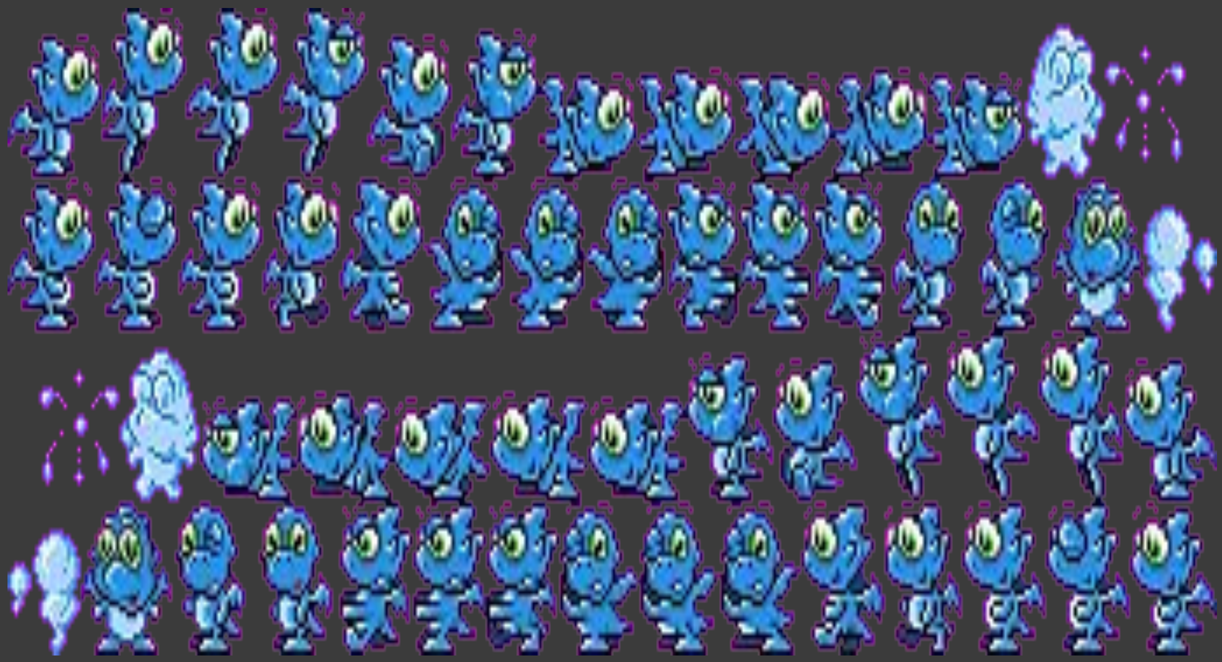
1) Spritesheet solution

- **<u>The most common solution:</u>**
  - we store each frame next to each other in a larger image

2) Separate image for each frame

- maybe processing is easier

Sample Spritesheet



Aladdin game (SNES)

Classic spritesheet

- **The animations are stored side by side**
- <u>Early spritesheet solution:</u>
  - Developer choose a uniform background color
    - so they know what not to display - **colorkey**
- Now we use alpha channel for this
- **<u>The size of the phases may vary:</u>**
  - during loading, the loader must be able to break it down by some logic,
    - then organize these sections into a separate texture
    - however, we need to know the dimensions of the frames
      - If the frames were in a separate file, that wouldn't be a problem
  - There is a need for some additional descriptive file!

Sprite descriptor file

- **What does a description file provide?**
  - Defines the frames exact pixel position
  - The exact size of the bounding boxes
    - a frame can have multiple boxes
  - Maybe the name of the frame
    - sometimes special frames need to be distinguished
  - it is possible to store any other data that is considered important
- **What format?**
  - What format?
  - It is advisable to choose a known storage format such as JSON or XML

A serious game needs some kind of descriptive!

Sample sprite descriptor file

```
<animation name="My sprite anim">
    <frames numofframes="2">
        <frame name="Robot Anim 1" file="robot1.tga">
            <aabb minx="0" miny="0" maxx="64" maxy="64" />
        </frame>
        <frame name="Robot Anim 2" file="robot2.tga">
            <aabb minx="0" miny="0" maxx="70" maxy="60" />
        </frame>
    </frames>
</animation>
```

Sprite based animation

- These kind of two-dimensional drawings are referred to collectively as "**Pixel Art**"
- **Reason:** Mostly drawn pixel by pixel
  - it's a difficult, time-consuming process
- Today, most games made for mobile devices fall into this category

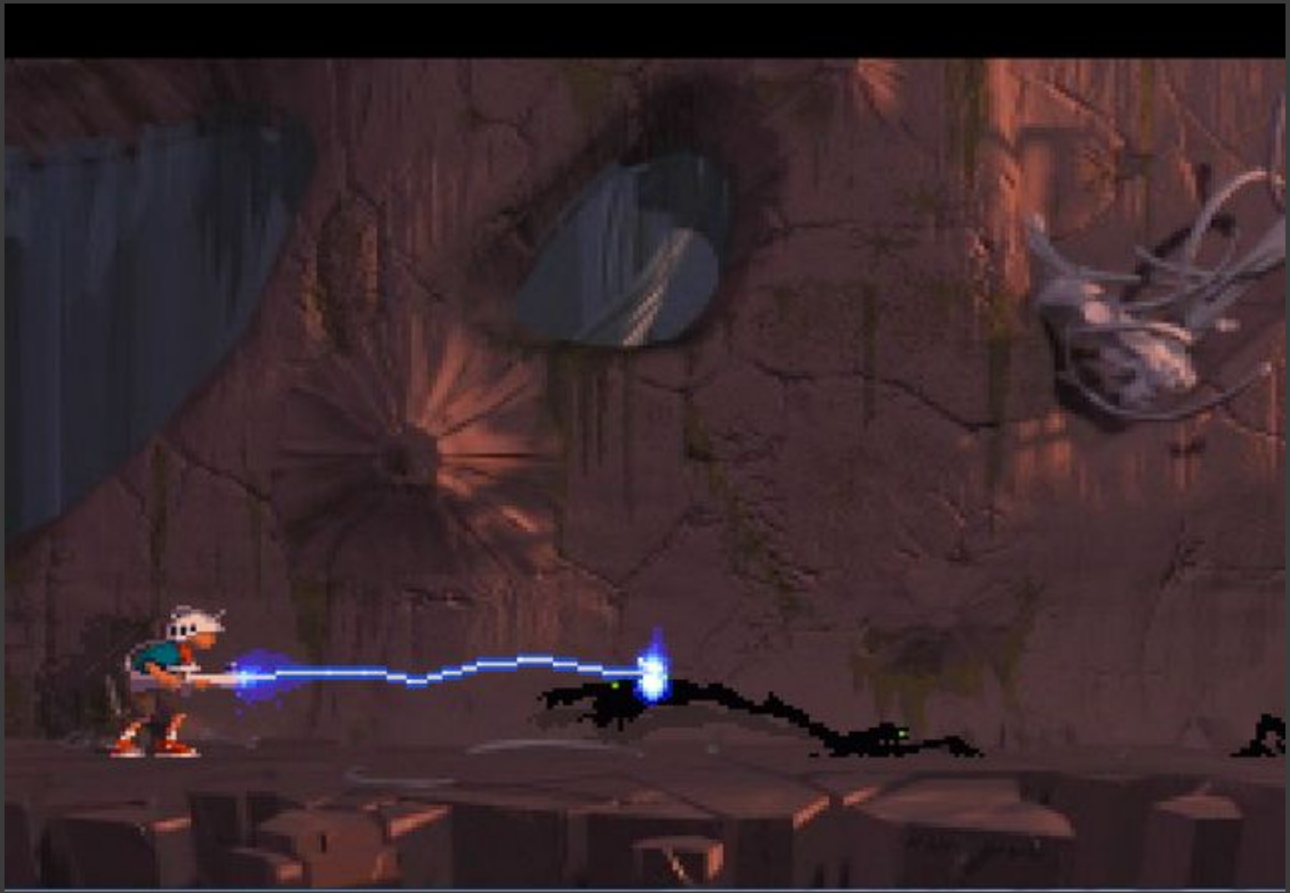
Sprite based animation

- The technique is also capable of producing very high quality so-called **cinematic** games
- **Key features:**
  - very smooth animation
  - many, even hundreds of frames
  - Animations can also be digitized:
    - Name: Rotoscoping

Famous Games:

Prince of Persia (1989), Flashback (1992), Aladdin (1993), Lion King (1994), Heart of Darknes (1998)

Heart of Darkness (1998)



Professional work:

- lots of cinematic elements
- lots of frames, smooth animation

Drawing the animations…

Drawing the animations

- The realization of the animation is to draw the various frames one after the other
- The speed of animation should be taken into account
- We cannot draw the next frame in each main loop
  - the animation will be too fast
- **What we need:**
  - to set the animation speed and consider it in the drawing process
- To achieve this, elapsed time can be used again!

Drawing the animations

```
/** Update frames */
void Update() {
    long ticks = GetOSTicks();
    // Decide to jump to next frame or not
    if ( 1000.0f/mFps < (ticks - mLastUpdate) ){
        mLastUpdate = ticks;
        if (++mActualFrame > mNumFrames){
            mActualFrame = 0;
        }
    }
}
```

Drawing the animations

Example explanation:

- *mFps* is the speed of the frame change

- The solution logic is simple:
  - *The value of 1000.0f / mFps* gives how many times you need to make the frame change in 1 second
  - When the elapsed time exceeds this value, we can switch to the next phase.

The GAMEOBJECT class…

GameObject class

- The **Sprite** class alone is not enough!
- **It can be used:**
  - For example as a basis for creating GUI elements (eg Animated buttons, etc.) or for actual game objects
- **Sprite is not complete in itself:**
  - In a two-dimensional game, an object has usually more than one animation
    - For different object state
- **GameObject:** an array of Sprites
  - where they can be changed depending on the state of the object (walking, squatting, etc.)
- We can call it **GameObject2D**

GameObject class

- The order the objects are drawn is important!
  - In some situations, objects may overlap each other
  - For example, there are objects (e.g.: Cloud) that are drawn on another objects
- The order is always based on the program logic
  - Level design question
- **Implementation:** requires the introduction of a numeric value
  - the order will be represented by this value
  - E.g.: *z* value

GameObject class

An implementation logic can be:

- The lower the z value of the object, the closer it is to the viewer,
  - this means that it will be drawn later
- The implementation requires sorting objects by their z-value
  - this ensures the proper order of the drawing

Sample implementation

```
class CGameObject2D {
    Vector2           m_vPosition;           // Position of the object
    Vector2           m_vNewPosition;        // Position of the object
    vector<CSprite>   m_Animations;          // Animation
    Vector2           m_vDirection;          // Direction of the movement
    float             m_fSpeed;              // Speed of the object
    bool              m_bVisible;            // Visible or not
    bool              m_bCollidable;         // Collidable or not
    int               m_uiCurrentAnim;       // Current Animation Frame
    int               m_uiNumberOfFrames;    // Number of Animations
    int               ID;                    // ID of the Object
    int               m_iZindex;             // z index of the object
public:
...
};
```

GAME OVER