CSD1130 Game Implementation Techniques

LECTURE

GAME ENGINE – SETUP (FOR CSD1450 & BEYOND)

Overview

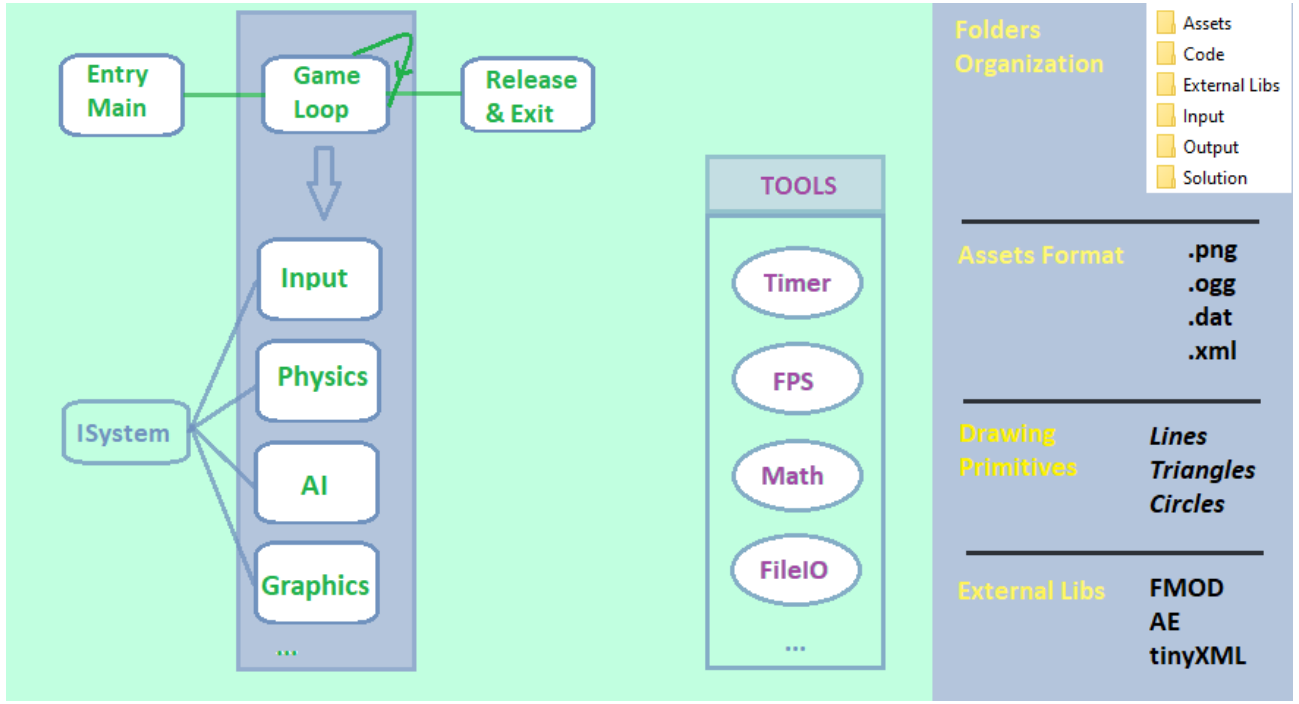
- Preparation
- Game Engine Architecture
  - Overview
  - Basic Setup
- Appendix – A
- Appendix – B
- Appendix – C

Preparation (1/4)

- *After forming your team*
  - *Get AlphaEngine lib.*
  - *Integrate with MSVS*
    - *Link and build in Debug and Release modes*
    - *Target: X64*
  - *Setup SVN/GIT team repository.*

Preparation (2/4)

- *After Setting up project environment*
  - *Start with a basic flowchart (or diagram) on your engine structure*
    - *This will be part of your TDD*
    - *What are the main systems, components, resources, libraries needed…*



Preparation (3/4)

- *After Setting up project environment*
  - *Decide on language*
    - *C++ is allowed*
  - *Decide on additional tools*
    - *Basic level editor*
    - *You can only decide on this, whenever you have a strong game idea direction*

Preparation (4/4)

- *After decisions are made on technical*
  - *Decide on team members responsibilities*
    - *Individual responsibilities*
    - *Based on interest*
    - *Based on needs*
    - *Based on upcoming milestone rubrics*
  - *Improve/prepare individual skills*
    - *Understand your tasks*
    - *Self learns on libraries usage (i.e. AlphaEngine)*
      - *Examples: How to read delta time, How to use MATH*
    - *Give ample time for research and make it quick!*

Remarks (1/2)

- *Get familiar with AlphaEngine*
  - *Play around with the given tutorial, in CSD1450.*
  - *Try new functions (APIs) that you can find in header files.*
  - *Ask your teachers, TAs…*
  - *In CSD1130, we'll see a lot of examples on AE usage, through our assignments.*
    - *That's why we ask for individual work!*

Remarks (2/2)

- *On Self skills*
  - *Students that have prior experience, can also help you!*
  - *Reading architecture books is great*
    - *But don't forget you have very limited amount of time!*
    - *Read whenever you are done with your tasks.*
    - *Mainly, follow your courses instructions*
      - *They are created to build upon.*
    - *Architecture will always be improved in the future!*
  - *Make your engine work first*
    - *Optimize later!*
    - *Optimization is done, only, when needed!*

Game Engine Architecture – Overview

- *How to start?*
  - *Ask yourself:*
    - *What are the needed systems?*
    - *How to organize these systems?*
      - *In visual studio project? In folders on hard drive?*
    - *What is a CORE engine?*
    - *Where is the program's entry point?*
    - *Where do I setup/link my engine's libraries?*
    - *What about the level's runtime? Level's switch?*
    - *What additional tools do I need?*
    - *How to handle my external assets?*
    - *How to represent the essential Lego pieces of my game?*

Game Engine Architecture – Basic Setup

- *What are the needed systems?*
  - *A list of logical and hardware systems*
    - *Also known as Managers.*
  - *These can be: Logic, AI, Physics, Collision, Graphics, Audio, Input, GSM,…*

Game Engine Architecture – Basic Setup

- *How to organize these systems?*
  - *In visual studio project?*
    - *Each system is a structure or a class.*
    - *Can have the following APIs:*
      - *Initialize*
      - *Update*
      - *Terminate*
    - *A system is responsible on its own actions.*
    - *A single/unique instance is needed.*
    - *They are globally visible*
      - *Especially from your CORE engine code files*

Game Engine Architecture – Basic Setup

- *How to organize these systems?*
  - *In folders on hard drive?*
    - *Each system has its own set of code files*
      - *In example: Physics system has "Physics.h" and "Physics.cpp".*

Game Engine Architecture – Basic Setup

- *What is a CORE engine?*
  - A CORE engine is seen as the manager of all the systems
    - Decides on systems organization/order.
    - Calls their APIs.
    - Is used in the Entry point code file.

Game Engine Architecture – Basic Setup

- *What is a CORE engine?*

```
struct/class CoreEngine
{
    void Init();
    void Update();
    void Exit();
};

//this function is called only one time before GSM loop and game loop
void CoreEngine::Init()
{
    //to initialize all the systems (following a specific order)
    //example:
        //Input.Initialize();
        //Physics.Initialize();
}

//this function wraps the GSM loop and  game loop
void CoreEngine::Update()
{
    while(...)
    {
        while(...)
        {
            //to update all the systems (following a specific order)
            //example:
                //Input.Update();
                //Physics.Update();
        }
    }
}

//this function is called one time after exiting the GSM loop and game loop
void CoreEngine::Exit()
{
    //to terminate all the systems (following a reverse order)
    //example:
        //Physics.Terminate();
        //Input.Terminate();
}

int APIENTRY wWinMain(...)
{
    CoreEngine ce;

    ce.Init();
    ce.Update();
    ce.Exit();
}
```

Game Engine Architecture – Basic Setup

- *Where is the program's entry point?*
  - In our case, it is the "WinMain".
  - Can be under "Main.cpp" file.
  - That is what we saw, so far:
    - Main loop
    - FPS control
    - GSM integration (as another wrapper loop)
  - Do integrate in your game engine!

Game Engine Architecture – Basic Setup

- *Where do I setup/link my engine's libraries?*
  - Your first library to integrate is "AlphaEngine".
  - You have learned how to integrate it in MSVS.
  - Do the same for additional libraries
    - i.e. FMOD
- Remark
  - Prioritize well on systems integration.
  - i.e. You only need FMOD after Week 7, unless your game is Musical!
  - Target only, your next milestone rubrics

Game Engine Architecture – Basic Setup

- *What about the level's runtime? Level's switch?*
  - The answer is given above, with GSM integration in the main loop.
  - Give each level its "Level_X.h" and "Level_X.cpp" files.
  - Each level is responsible on its own:
    - Logic
    - Assets
  - As an example, in a Platformer game, a level can be from 2 and up to 3 minutes of game play, on average.

Game Engine Architecture – Basic Setup

- *What additional tools do I need?*
  - *There are various of engine tools that you may add:*
    - *MATH (AE has one)*
    - *FPS*
    - *TIMER*
    - *Memory Manager*
    - *Resources Manager*
    - *FileIO*
    - *…*
  - *They have their own separate headers and code files*

Game Engine Architecture – Basic Setup

- *How to handle my external assets?*
  - *Answer is, through an Assets Manager:*
    - *Can use libraries to open/close specific file formats.*
    - *Can store assets*
      - *.PNG files, .OGG files, .JSON files…*
    - *Assets can be shared by multiple levels (Game States).*
    - *It is used by the GSM, Game State's Load/Unload APIs.*

Game Engine Architecture – Basic Setup

- *How to represent the essential Lego pieces of my game?*
- *In other words, what makes a game alive?*
  - *Answer: Game Objects!*
    - *Aka: Entities*

Game Engine Architecture – Basic Setup

- *GAME OBJECT*
  - *Can be a structure or class*
  - *Holds mainly*
    - *Data*
    - *Logic/Behavior*
      - *Can be as basic as "Init","Update","End" functions.*

Game Engine Architecture – Basic Setup

- *GAME OBJECT – Data*
  - *Position, Orientation, Scale.*
  - *Mesh (can be an ID). "Normalized original mesh?!"*
  - *Texture (can be an ID)*
    - *Or a color (as a modulation)*
    - *Or a list of Animations (where an animation is a list of frames)*
  - *Physical Properties*
    - *Mass, Velocity, Acceleration, Forces, …*
  - *Collision Properties*
    - *Shape, Data, …*

Game Engine Architecture – Basic Setup

- *GAME OBJECT – Data*

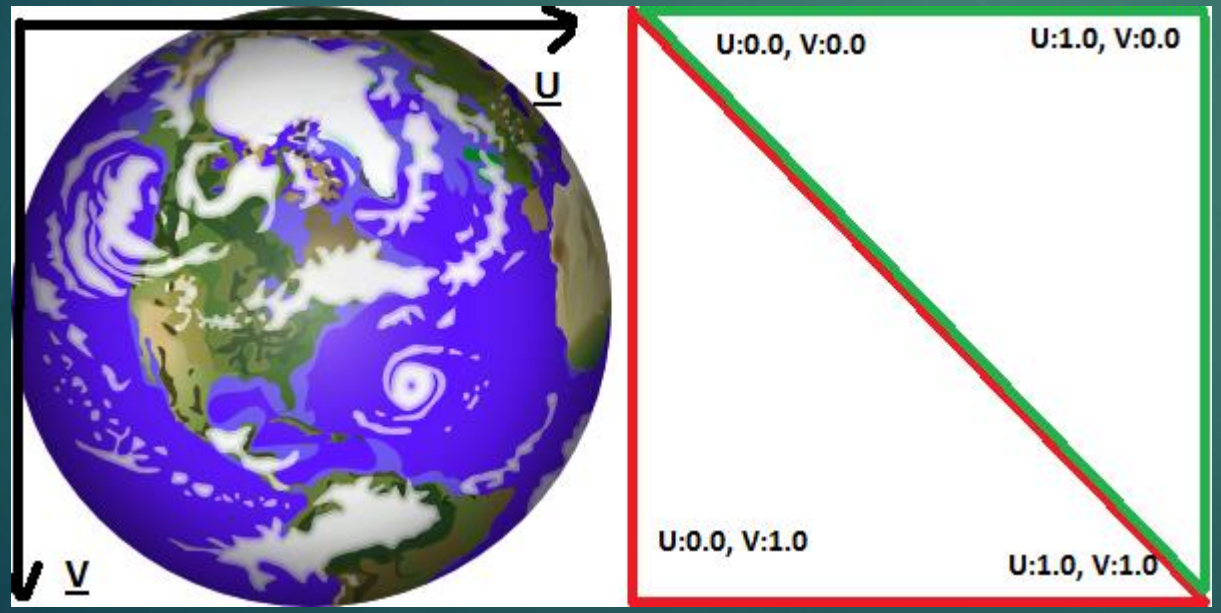
```
struct GameObject
{
    m_position -> (data for translation matrix)
    m_orientationAngle -> (data for rotation matrix)
    m_scale -> (data for scale matrix)
    ...

    m_mesh * (can be shared)
    m_animation(s) * (can be shared)
    ...

    m_mass
    m_velocity
    m_acceleration
    m_friction
    m_force...
    ...

    m_collisionShape (type)
    m_collisionData e.g. AABB, Circle...
    ...

    (*Init)
    (*UpdateLogic(GameObject *))
    (*End)
    ...
};
```

Game Engine Architecture – Basic Setup

- *GAME OBJECT*
  - *From the above structure, we learn that data must be updated.*
  - *Data updates are in their correspondent systems*
    - *i.e. Animations will be updated under "Animation" sub-system of the "Graphics" system*
    - *i.e. Mesh will be update in a "Transformations" system*
      - *Transformation can be a system by itself, to compute the final position, scale and orientation of an object's mesh*

Appendix – A

- *Texture Mapping*
  - *A texture, like a .png file, must map its UV coordinates onto a mesh's vertices, for it to be rendered.*




```
AEGfxMeshStart();

// This shape has 2 triangles
AEGfxTriAdd(
        -30.0f, -30.0f, 0x00FF00FF, 0.0f, 1.0f,
        30.0f, -30.0f, 0x00FFFF00, 1.0f, 1.0f,
        -30.0f, 30.0f, 0x0000FFFF, 0.0f, 0.0f);

AEGfxTriAdd(
        30.0f, -30.0f, 0x00FFFFFF, 1.0f, 1.0f,
        30.0f, 30.0f, 0x00FFFFFF, 1.0f, 0.0f,
        -30.0f, 30.0f, 0x00FFFFFF, 0.0f, 0.0f);

pMesh2 = AEGfxMeshEnd();
```

In Red are the UV coordinates

Appendix – B

- 2D Basic Animation
  - A series of frames (pictures of the same size).
  - Uses a single mesh (2 triangles to form a quad).
  - Can use a series of separate pictures, or one picture that holds all the frames
    - Called Sprite_Sheet.

Appendix – B

- 2D Basic Animation – Sprite_Sheet
  - Basic code structure:
    - Frame Structure
      - U, V coordinates (floats)
      - Time delay (in milliseconds)
    - Animation Structure
      - List of Frames
      - Width & Height of a frame

A farmer Front Walk

Appendix – C

- *Game Objects differ by their jobs (behavior in the game)*
  - *Dynamic objects vs Static objects*
  - *Camera object*
  - *Viewport object*
  - *Audio object*
  - *Text/Font object*
  - *UI object*
  - *Particle system object*
  - *…*