Project Proposal (Colonization I)

Jiaying Ye, j33ye

November 2015

The purpose of this project is to create a city builder/resource management game with as many gaming graphical features as possible.

The scenario of this game is that the player is managing a space colonization program on the surface of another planet, whose main goal is to collect and utilize the resource in order to survive and extend the base of colonization.

1 Topics:

- texture mapping on various meshes
- fog in distance
- time-based lighting sources
- sky-box background
- Animations
- Multi-threading
- Transparent material to make glasses on buildings
- User events handling
- Frame rate optimizations
- double buffering

2 Statement

- **Layout:**
  - 2D Status hub, to display resource and status, game menu, etc
  - A limited area of surface to build on, rotatable and zoom-able camera

  - A reactor is placed in the center of the map served as the core of the base, all other buildings are linked directly or indirectly to the core by pipes.

- **Main game flows:**

  - save/load game status
  - time changes, sun rises and sets resulting in different lighting and shadow
  - mouse picking/dragging building from panels into scene

3 Technical Outline

- **Control flows, Game events, Multi-threading:**
  MVC pattern is used here. View being as the window, Model being as gameData, Controller being as gameControl. These are three separated threads to achieved improved performance. The way to communicate between them is to pass "gameEvent", where the event type and information is stored. "gameEvent"s are stored inside a mutual exclusive queue, and processed by its thread accordingly.
  More specially, rendering type "gameEvent" is past to View(window), a simple technique is used to achieve a fixed FPS, that is timing each processing runtime and process as many "gameEvent"s as possible between the time of two frames.
  Event loop inside others(Model, Controller) can be implemented as busy-waiting on its own event queue, this might melt the CPU, but it seems the OS is doing a good job on preempting the busy thread. If it does waste too much resource, this can be fixed by using a counting semaphore on the queue.

- **Data storage:**
  Files(models, save, game scenario definition) are stored as simple INI files for easy modification and access. For now, it seems over-killed to introduce storage like XML, but this can be a backup plan.

- **Distance fog:**
  Since the model of the planet surface is limited size, it looks bad at distance. Putting some fog in the far side can make the game looks much nicer and more realistic. This can be achieved by writing a fog shader and more fog is put on when the distance to camera is larger.

- **Texture:**
  UV texture mapping is used. So the process of generating of texture can be split into other software (blender)

4 Bibliography

Adding some libraries into the project to save time on basic operations:

- Assimp, "http://assimp.sourceforge.net/"
  for loading all types of formats of 3D models
- C++ Boost, "http://www.boost.org/"
  for simplified multi-threading. It also provides some help in networking.
- free 3D model sites, "http://tf3dm.com/"
  for downloading nice and free 3D models
- Pixel City - Procedurally generated city, "http://www.shamusyoung.com/twentysidedtale/?p=2940"
  A project about generating different shape of city buildings. It talks about texture mapping and lighting. The fact that this is using such simple model and texture to achieve great result makes me believe that I can learn a lot to build my own project.

5 Objectives

□Game can be saved and loaded

A city builder game should be able to save the the layout of the city, states(levels) of every building and amount of resources.
Since all these information of game state is very simple and can be expressed by integers and strings, storing save in a INI file is nice for reading/writing data. Although I can include some fancier file format, like XML, this will probably be over-killed with such a small amount of data.
Along side with saving game state. A set of INI files under *GameData* folder are also used to define the meshes of in-game models, sky-box, building surface, etc. This can save some time and also allow different and interesting building surfaces to be in the game.

□Texture mapping on buildings and surface

By including *Assimp* library, it enables the game to load more 3D model formats than just .obj. Also, it reads UV from the model, so it is nice and simple to read and pass UV to shader and do the texturing job outside coding (in Blender, etc).

□A core of the city is renderred and buildings are connected by pipes

As the most basic models of this game, a reactor model is severed as the core of the city. Pipes are procedurally generated with the coordinates of two connecting buildings and connects two buildings.

□A time-based sky-box background

Sun sets/rises as time goes by, so it is necessary to change to color of the sky as well. The scenario of this game is colonizing another planet, maybe a martian-ish sky would be suitable. By "adding" color to the sky, it goes darker like night times. If there is more time, I can also add procedurally generated star maps on the sky to make it look nicer.

□A 2D Status Hub

It might be complicated to implement a full text supported 2D hub, but at least a hub to display player's resources is necessary. This can be a textured progress bar on the screen. Without such a hub, the game would be hard to understand.

□Basic sound engine

Simple sound is played for certain event, like upgrading a building. Also background music can make the game more atmospheric, given that animation is much difficult to make, at least adding sound is nice and affected.

□Fog in the distance of the surface

The size of the building surface is very limited, adding fog in distance (blend scene into sky color) makes the game look nicer.

□Building can be placed/upgraded on the surface with a grid-like layout

Grid surface map is perfect for a city builder game, because it allows easy placing/storing of the city layout. With this, game objects of building can be denoted with their unique coordinates.

□Time-based lighting is implemented so that the direction of light is moved in different time

The sun as a light source is moved along a fixed arc on the sky with a "day-night" cycle of 10-mins in real time to simulates the day-night light changes. Changes of sunlight color can also make the game more atmospheric.

□Shadow is correctly casted

Shadow mapping can be used here. This method of casting shadow is suitable for this game since player is watching the city far up, it doesn't need very detailed shadow. Shadow mapping can do that efficiently.