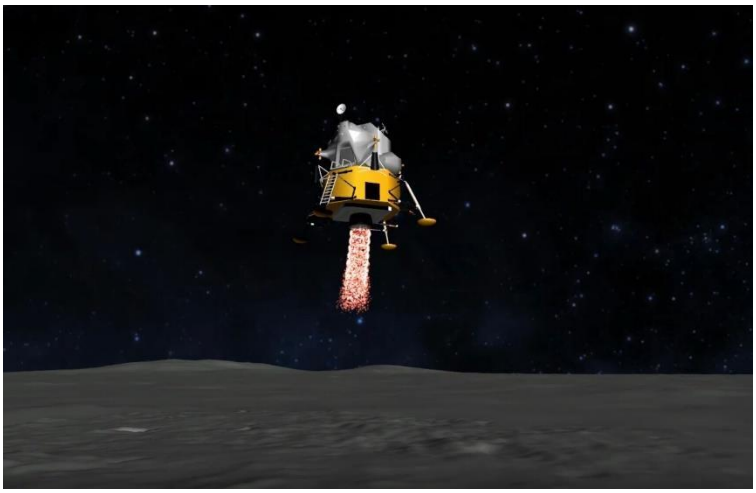
SJSU - CS 134 – Game Design Programming Project 3 – Final Project

3D Lander – Putting it all Together

In this final project, you will create a substantial portion of a 3D Lander-style game from scratch in C++ using the components you have worked on from pervious assignments including lighting, physics, octree subdivision, particle simulation and more. The player will be able move the "vehicle" using physics and thrust forces to control landing of the spacecraft on a highly detailed terrain model. The player will have control over the motion of the craft, camera views. Upon successful decent, the game will do collision detection between the craft and the surface of the planet while the player touches down. Accompanying sounds will enhance the experience. The example shown in class is a "Lunar Lander" style, but you can customize the experience to any type of vehicle-terrain interaction game if you meet the technical requirements.



Game Play

The player is given two minutes of fuel. Fuel is exhausted only when the thrusters are on. When the fuel is exhausted, the thrusters no longer operate. The amount (in time) of unexpended fuel must be displayed in viewport. The LEM pilot must land the spacecraft (in moon gravity) in the designated high-resolution landing area on the 3D terrain. If the landing is too hard (as measured by the contact forces), the spacecraft will explode (a 3D explosion will catapult the ship away uncontrollably). This will signify that the game is over. If the ship lands safely in the landing zone, the player wins.

Required Functionality

The required technical requirements for the project is as follows:

- The LEM must be able to maneuver (left, right, forward, back, up/down and rotate along it's UP axis) using thrust force using a **full physics simulation** (which includes thrust, gravity and turbulence forces.) This was covered completely in the midterm for the 2D case. You need to add another dimension. A non-physics way of moving or a hacked physics implementation will not receive credit. Students can choose to write a custom simulator as we did on the midterm or use the particle simulator example. Since the particle simulator supports external forces, this may be an easier starting point.

- The craft will be equipped with a telemetry sensor to detect altitude (AGL) above ground level and display it on the screen. You must use ray-based collision detection to accomplish this ask. The ray must intersect the 3D terrain. The feature should able to be turned on/off with a hotkey.
- The ship must use a particle emitter for the rocket exhaust. A particle emitter must be used for the explosion.
- Particles must be rendered using a shader otherwise it will be too slow. An example will be given in class.
- The ship must be able to do full collision detection with the terrain (including mountains and landing area). If the ship contacts the terrain, the collision must be resolved with a suitable impulse force that can be demonstrated. You are only required to have one (1) bounding box on your vehicle, but your terrain should be spatially partitioned using an octree. Your demo video must show collision with the landing area AND mountainous terrain to prove that it is reliable.
- Use lighting techniques learned in class to light the landing area scene using OpenGL lighting (ofLight) using at least three (3) lights. Adding an additional light on the spacecraft that can be turned on/off is an interesting effect.
- The player should be able to switch between multiple camera views which include a minimum of:
    - ○ A "tracking" camera that stays aimed at the spacecraft from a fixed location.
    - ○ One onboard. Can be aimed in different directions, depending on your scenario.
    - The player will also have to have access to the EasyCam camera to be able to navigate anywhere they like over the surface.
    - The EasyCam camera should have an option so that you can retarget the view to a point or the spacecraft.
- Sound is required for effects (spacecraft thrust) and any background sounds you would like.
- For diagnostic purposes, you should have a feature (such as in the) to be able to manually move the spacecraft around the scene using the mouse directly and then start your game from that position.
- The scene should have a simple 2D image background (such a starfield).
- Your game trailer will be graded as part of the project and must show all features. If a feature is missing from the trailer, we will assume that it is not implemented and it won't be credited.
- ~~All code and assets required to build and run the game must be submitted. If your game doesn't compile and run or features that are shown in the video do not match the source provided, you will not receive credit for the project.~~
- *Here is a movie demonstrating most requirements:* https://youtu.be/QSbsvQw2ljQ

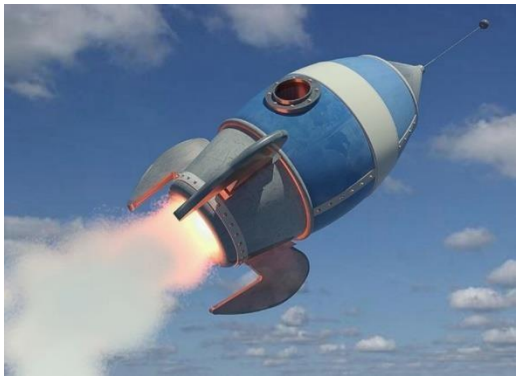
Suggested Steps/Hints

Even though the project seems complex, you have already completed most of the technology and R+D needed for the game:

- Consider using simple particle motion to move the spacecraft and add an additional integrator for rotation in "Y".
- Use the class particle system to generate a thruster simulation.
- Use the octree code you developed for altitude tracking and collision detection.
- Use the octree code for any 3D selection you may want to do on the surface.
- ***It will be required to exercise wise time management to complete the project with a good level of quality.***

Substituting Content

If you are interested in substituting content and creating your own "theme" of a game, you can but it must meet the same technical requirements.



- Create your own "vehicle" and/or terrain in Maya (or modeler of your choice). If you choice to use an existing model, copyright laws apply.
- **Terrain must be at least 200K vertices and contain interesting features. (not just flat plane).**
- still required to provide a custom emitter that produces a exhaust-like effect of your choice even if the vehicle of your choice doesn't have a rocket engine.
- Source for example image at right: https://www.pinterest.com/mcgo0087/retro-rockets/?lp=true

What to Submit

1. **Submit the game code in canvas. Do not include your trailer in the .zip file. You must sign each module or function that you develop in the code.** Source must be submitted in the *<Name>_Project3_<date>.zip* format.
2. *You must submit a trailer to the link to a gallery to be provided and to the Canvas assignment.*. The trailer must be a screen capture with sound with **no product watermarks**. No screener movies with an phone camera will be accepted. The trailer should be one that you are proud of and can refer to in your portfolio or project listing on your CV.

Grading Criteria

A rubric will be provided in the assignment on Canvas. Grading will be passed on the list of requirements in this document. The trailer will also be included in the grading (important).

Collaboration

Students can ask questions on the Discussion section of the class in Canvas, share ideas and programming information, but you cannot share source code. Each student must create and submit their own unique solution to the game.