PRESS THE

HOME

BUTTON

TO START THE GAME

&

DISMISS THE INTRO

SCREENS

[HOME = Motherland]

Soviet Rush

2nd Submission v2.0

What we implemented:

- We implemented a wavefront .obj model loader with a custom texture binder and loader.
- We have a Terrain with texture
- We have lots of 3D models in the world
- We have a grid for determining game logic and events
- The grid contains cells for micromanaging of the objects.
- Each cell contains a custom implemented double linked list with all(if any) models.
- Models/Objects transfer automatically from one cell to another cell upon movement or border crossing(simple collision and border detection)
- We have implemented custom loaders for two types of images – BMP and TGA.
- Skybox with a nuke – TZAR BOMB, always centered around the camera when the player moves in any direction.
- Textures for the 3D models, plane, world, skybox etc.
- We have MP3 Playback using the FMOD library, with the feature of changing songs(from a playlist) and muting them (if one does not like listening to classical remakes of soviet folk songs)
- There are also some sound effects like the engine roar when accelerating and a powerup notification.
- We have implemented a custom FPS counter like no other with a string representation of the numbers.
- We have implemented the following list of special functions required by the CG gurus:
    - F1 - Help (if available)
    - F2 - Framerate on/off
    - F3 - Wire Frame on/off
    - F4 - Textur-Sampling-Quality: Nearest Neighbor/Bilinear
    - F5 - Mip Mapping-Quality: Off/Nearest Neighbor/Linear
    - F6 – Toggle Fog on/off (additional feature adds to the depressive soviet
    - atmosphere)
    - F8 - Viewfrustum-Culling on/off
    - F9 - Transparency on/off
- We have implemented a custom heightmap loader which determines the height of a given point on the plane (X,Z) depending on the gray scale value of it. (Using a custom TGA loader)
- We have 2 different directional light sources – yellow from the back side (where the sunset should be) and reddish from the bomb.
- We also have implemented a simple user interface or HUD which shows the current life/boost/ammo the player has and updates automatically upon a collision with a powerup.
- We have randomly generated powerups of different types all over the plane
- Our main player is a complex object containing 3 different kinds of meshes – the unicorns body and the 2 wheels. The wheels are animated using opengl/c++ upon movement – backwards and forwards respectively.

- We have implemented collision detection using bounding spheres for the player and each powerup or obstacle object.
- We have custom made help screens and gameover / victory screen which should bring joy to the user!
- Our camera has 2 fixed positions on the unicorn which the user can change using the 'C' button. The camera is always fixed a little behind the unicorns position and follows it.
- We have implemented a Vertex and Fragment shader – it's a simple Phong Shader which works with multiple light sources.
- We also have a simple Texture mapping vertex and fragment shader(only 2D Textures).

Implementation of the requirements:

Gameplay

For our gameplay we have implemented a simple racing game with a unicorn instead of a car. We have implemented simple collision detection against boundaries / walls and power-ups in order to make the game feel more 'realistic' to the player.
The goal of the game is to get from the start to the finish without dying – getting bad powerups or crashing. If the player has managed to get to the finish without crashing or dying a “Victorious” screen pops up notifying that the user has won the game. In the other case if a player crashes into a wall or collects to many bad powerups then the game is over and a screen pops up notifying that the user has failed his comrades. We wanted to make the game fun and easy to play so we can entertain the user – hopefully we have achieved this.
How we implemented:

Model-loading

We have wrote a custom .obj loader so we can load some 3d models into our game and make it look more interesting and interactive
Collision detection

We have implemented simple collision detections using bounding spheres where each object has a radius. We check for collision in the cells between the objects that are there using the following formula:

```
if(object1.radius + object2.radius <=
distanceBetween(object1,object2))
{
//collision occurred render some stuff
}
else {
/* nothing to do here
move along */
}
```

HUD /Victorious / Game over screens
Powerups
-We have achieved drawing an user interface and the victorious / game over screens through switching from a GL_MODELVIEW perspective to an orthographic projection where we render the user interface and screens. After rendering all we need in the

orthographic projection we quickly switch back to our GL_MOVELVIEW matrix so we can render 3D again. So our game loop looks a bit like the following:

```
//In GL_MODELVIEW
check what has changed since
past frame
switch to ortho and render
UI
switch back to GL_MODELVIEW
render some 3d stuff
```

Our powerups are 3D objects with different attributes, depending on what kind of powerup is generated. They are generated with random attributes and at a random position at the start of each game.

Boundaries checking

-We use a very simple 'naïve' boundary check. We check each moving objects coordinates against the world boundaries if they are near (e.g. if they are in a cell that is a border cell). This way we can model the game behaviour better – no objects fly away into the infinite nothingness of space beyond the viewing frustum. We basically just check the (X,Z) coordinates of an object against either the X or the Z coordinate of a cell.

Illumination of the objects

We have a very simple illumination effect in our game. We have 2 directional light sources and an ambient light throughout the whole world. All objects on the plane are subject to this illumination – it's sort of a 'global' illumination. All objects are textured – for that we use the simple texturing functions opengl provides – generating , binding, applying – and two custom image loaders which load the images / textures into memory.

Additional libraries

FMOD

We use the FMOD library for mp3 playback.

GLM

We have tried a lot of different object loader libraries – like Assimp, DevIL for .dae (collada), also tried diverse loaders for .3ds and other formats none seemed to work or lacked documentation. We stopped at GLM for loading more simple objects like wavefront .obj and modified it so that it would work well with our textures and objects.

Implemented Effects

The only effect we managed to implement is a simple fog simulation using a build in opengl function. Due to hardware restrictions and time limitation we did not succeed in implementing the effects we wanted to.

Other special features

Other special features in our game are mp3 playback with a specially selected play list of classic soviet folk songs with a taste of rock 'n' roll.

3D Modelling Tools

The tools we used to model our objects are Google's Sketchup since we have no prior knowledge with more complex programs like 3D Studio Max or Maya.

Game Walkthrough

The object of the game is simple – as a player you have to get from the start to the finish line of the racing game without crashing or getting killed by radioactive mushrooms. You can navigate the unicorn with the following keys:

- W - accelerate
- S - break / reverse gear
- A - turn left
- D - turn right
- LMB (Left mouse button) to shoot
- RMB (Right mouse button) to boost

What we are going to work on:

- Particle system / effects
- Motion blur while boost
- Fix camera / unicorn position issues