CG Assignment

By Kei Sum Wang
19126089

Algorithms

Functions

Collision detection(obj_near(vec3, nearDist)) - collision detection was implemented in a very simple way, requiring only an object's current position and current camera position to calculate the distance.
In the source code, this was implemented as a function called "obj_near(vec3, nearDist)" where the vec3 parameter is a position and nearDist is the distance that you consider to be near. This was used throughout the program for situations such as picking up the light source and sven when nearby, and for player defeat when in contact with an enemy AI.

Jumping functionality- To implement jumping, the following functions were created:
jump_rise(void) which handles the rise when pressing the jump button by incrementing the y position by 0.1 until reaches peak of at which is determined by the difference between the last frame and the frame the jump button was pressed and once the peak is reached it signals the down_gravity function. The reason for basing the peak height on time is because this allows more consistent results when performing the jump and prevent teleporting, which are issues that are influenced by hardware capabilities.
down_gravity() keeps the camera grounded to prevent flying when moving around keeping the camera position at 0.9 y axis which is the default y position of the camera implemented .
jump_rise() also helps this function determine whether the camera needs to descend after jumping.

Walking animation function - walk_animation(float rotDelta, int obj_pos). rotDelta is the scalar used to increase or reduce the speed of the rotation animation, obj_pos is the position of a model.
Mathematics behind creating is using the sine function and program time to calculate the angle for rotation. Sine is an appropriate math function to use since it creates a pendulum swing movement, sine does this by taking an angle(time) and calculates the position of where the angle would be on the sine graph which -1 to 1, when iterating the sine the result will go from -1 to 1 and vice versa creating swing animation. Water sheep and the skeletons use this function to display walking animations

Sheep_mover() - This function helps determine the speed of sheep's movement towards the camera by incrementing the scalar value which is used by the unit vector between sheep and camera. Function also determine if the player is killed by the sheep.

Quest_started - checks if the player is towards the objective location and away from spawn.

Modelling algorithms

Modeling mainly consisted of storing 3D positions into 3D vector arrays, this allowed details of models to be grouped into one object which allowed for more efficient and less messy modelling algorithms. For loops were mainly used to access each detail of contents stored in the vector array, at this point texture binding and matrix transformations are performed to form and connect the composite objects when rendering.

Light source modelling - Light source modelled represents a lightsaber, which has 2 parts, the hilt and the light source itself. Both parts are modelled using rectangles(stretched cubes) to form the complete object. Lightsaber is initial on the ground during each render until the player picks it up which is handled by collision detection. When Lightsaber is picked up by the player it is translated into desired position then multiplied by the inverse of the view matrix which allows the lightsaber to follow the camera without any rotation like any fps game. The inverse of view matrix is coordinate of the view in model/world space, this can be multiplied with any model to bring that model into the view position. The light source uses a fragment shader that does not handle lighting, this is to emphasise that source is really the light source by showing constant brightness of the object. The lightsaber also uses a different vertices array to fix and make texture positions more uniform, for example box with smiley faces facing and rotated the same position rather than different. Water sheep uses the learn opengl provide curtin specular since the curtin specular is shown to provide somewhat intense reflection, which is perfect for the lightsaber as both parts of the saber can be reflective.

Sven - Sven is a wolf that is modelled with 15 objects that based on the cube. These 15 objects include the face, mouth, eyes, ears, body, nose, legs, and tail. Sven starts laying down which is done by rotation by 90 degrees along the z axis. Picking up sven is done the same way as the lightsaber, by bringing sven to desired position then multiplying by the inverse view matrix to keep sven with the camera view. Sven uses the learn opengl provide grass specular since the grass specular is shown to dull, wolves realistically shouldn't really be reflecting light.

Water sheep- is a sheep modelled with 12 objects that are based on the cube. 12 objects include the body, parts of head, face and legs. Sheep starts of facing the negative x axis near sven. Sheeps position is affected when the player picks up Sven, chasing the player down. How the sheep chases down the player is done through the use of normal vectors between the camera and the sheep's current position, the sheep is translated by coordinates of the normal vector which are multiplied by scalar which determines the speed the sheep is moving at. While chasing the player, the shape also constantly face the camera positions, this is done by finding the angle of the sheep's normal vector direction to the camera position. The atan function is used to calculate the angle by taking the directions x and z values to determine the hypotenuse which in turn can also be used as an angle for rotation to keep the sheep facing towards the camera position. Water sheep can also kill the player if they are close, which done through collision detection. Water sheep uses the learn opengl provide grass specular since the grass specular is shown to dull, sheep realistically shouldn't really be reflecting light.

Skeleton- is the water sheeps minion that helps attack the player. Skeleton consists of 6 objects based on cubes which include the head, body, arms and legs. Skeletons do not spawn until sven is picked up. Once Sven is picked the skeletons will join water sheep in pursuit of the player, chasing also done in the same that implemented with water sheep using normal vectors between camera and the skeletons current position. However the skeletons have constant scalar value for the direction translation since they are designed to be faster than water sheep. Killing the player is done the same as water sheep through collision detection. Skeleton uses the learn opengl provided marble specular finish since I personally liked the idea of a metal skeleton.

Helicopter - is a model mainly for the animation effect. It comprises of 13 objects based on cubes which include the body, blades, rotors, landing gear, tail and the fin. The helicopter is the vehicle that “drops off the player” and takes off until the player has completed the task of saving Sven. Again flying and take off is done by the use of normal vectors between desired position and current helicopter position. Upon completion of bringing sven to the helipad, the helicopter will fly to the helipad and land. This is done by using the normal vector between the helipad and current helicopter position. Helicopter uses the provide curtin specular due to it displaying appropriate reflective shine qualities that realistically would occur on helicopters. The helicopter also utilizes shearing to create a sleek look on the body and tail, the coefficient used for shearing is the z axis since we want shear to the z axis as the helicopter is facing negative z axis.

Plants - 2 plant models are implemented which are the yucca tree(non trap) and the cactus(trap). Yucca tree consists 6 objects while cactus only uses one rectangle. Yucca trees are added for scenery and utilise shearing for the branches to help the model resemble the tree more realistically. Shearing done towards the y axis as we want the branches move up diagonally. Cactus are just rectangular plants used for traps when player picks up sven, collision detection is used to determine if the player is killed by a cactus. Both plants are scattered through the scene using modulo algorithm, if current translation x and z value is divisible by 2 then divide x by 2 or multiply by 4 if not divisible, this add variability to the scene. Both use the learn opengl provided grass specular as plants are not to have intense reflective qualities.

Water - just a simple rectangular model that represents water for scenery leading to water sheep. This uses the curtin specular as water can be reflective when light source is near.

Lava - simple square that acts as a trap when player picks sven up. Uses the marble specular as lava material can reflective with its own light, however I didn’t implement the lava to be a light source.

Road and helipad - road is a simple rectangle that leads to a helipad that is a simple square. Uses the provided street specular as this specular is the most appropriate for representing due to the fact that they are roads

Huts - 2 models are the same huts with different textures. One has uses wood texture and specular while the other use brick texture and specular. The wood is slight but dull reflection while brick texture has a small but sharp reflection. Both textures are provided by learn opengl. The brick hut house sven and water sheep while the wooden one is for scenery and a spawn point for a skeleton.

REFERENCES

All textures were found on google images that reference to open source images and some of the textures were already provided with the Learn Opengl package.

Skeleton textures -
https://www.pngfind.com/mpng/ixRJwTo_wither-skeleton-skull-minecraft-hd-png-download/

Wolf texture - http://bgfons.com/download/1719

Lightsaber- https://pngimage.net/bob-minion-png-6/
https://www.pngfly.com/

Radioactive box-
https://www.pngguru.com/free-transparent-background-png-clipart-nftna

Cactus- https://myrealdomain.com/explore/cactus-texture.html

Lava -
http://www.textures4photoshop.com/tex/fire-and-smoke/lava-magma-seamless-texture-free-download.aspx

Helipad - https://www.pngfly.com/free-png/helipad.html

Grass -
http://www.monkeymods.com/fallout-4/modders-resources-and-tutorials/jesters-seamless-texture-pack-modders-resource-for-fallout-4/attachment/jesters-seamless-texture-pack-modders-resource-for-fallout-4-17/

Yucca tree - https://pxhere.com/en/photo/966522

Tool used to customise some textures
-https://www7.lunapic.com
-gthumb on ubuntu