COMP 3011/GRA

Computer Graphics Coursework

Kai Wey, Lim

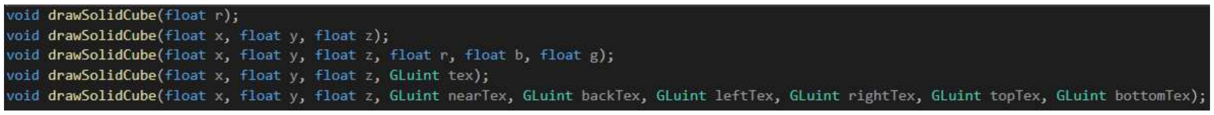
20011792

1.0 The scene

The setting chosen for the scene in this assignment is a village with a plain mood. The design of the village is heavily inspired from the game, Minecraft. Therefore, the design of the structures is relatable to the game. The entire scene is encapsulated within a skybox, drawn with the drawSkyBox() in the drawSkyBox.h, header file. The drawSkyBox() parameters are a float that defines the size, and 6 different textures for each face of the cube. This allows a different texture applied to each side of the cube that allows the user to easily differentiate which side of the skybox the user is currently positioned at.

2.0 The Models

2.1 Technique used to draw the models.



```
void drawSolidCube(float r);
void drawSolidCube(float x, float y, float z);
void drawSolidCube(float x, float y, float z, float r, float b, float g);
void drawSolidCube(float x, float y, float z, GLuint tex);
void drawSolidCube(float x, float y, float z, GLuint nearTex, GLuint backTex, GLuint leftTex, GLuint rightTex, GLuint topTex, GLuint bottomTex);
```

Figure 2.0: The drawSolidCube() variations

The models in the scene are largely made of cubes that is created with triangles through the function drawSolidCube(), in the drawCube.h, header file. There are a few variations of the drawSolidCube() that are used to draw different parts of the scene. The first drawSolidCube() as shown in Figure 2.0, allows the creation of a cube with the same length, width and height, while the second function allows the specification of the length, width and height for the cube, which in this case allows the drawing of cuboid. Both these functions draw a cube or cuboid with the default color of gray. The third drawSolidCube() allows the drawing of a cuboid with a specified color where the float r, b and g, changes the ambient and diffuse color of the cuboid. The fourth drawSolidCube(), allows the cuboid to be drawn with a specified texture which covers each face of the cuboid individually, instead of filling the cuboid with a color. The fifth and last drawSolidCube(), allows the cuboid to be drawn with a different texture on each side of the cuboid. This is achieved by initializing a different texture for each side of the cuboid. All the drawSolidCube() have their respective normal specified on each side of the cuboid to allow the cuboid to have a reflective surface when light is shined on it.

Subdivision is also added to the scene by subdividing an octahedron. Each side of the octahedron is made of a triangle which is subdivided until the specified depth, to form a sphere like shape. Through subdividing, the octahedron will gradually form a sphere as the subdivision depth increases. The functions used for subdivision are placed in drawSun.h, header file. The function that is used to draw the subdivided octahedron is the draw(), where inside consists of the subdivision function, divide_triangle() for each face of the octachedron.

2.2 Models Found in the scene

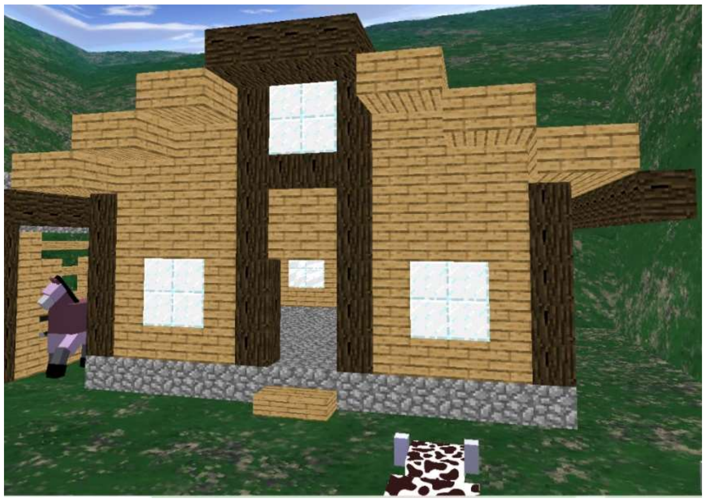


Figure 2.1: The house model.

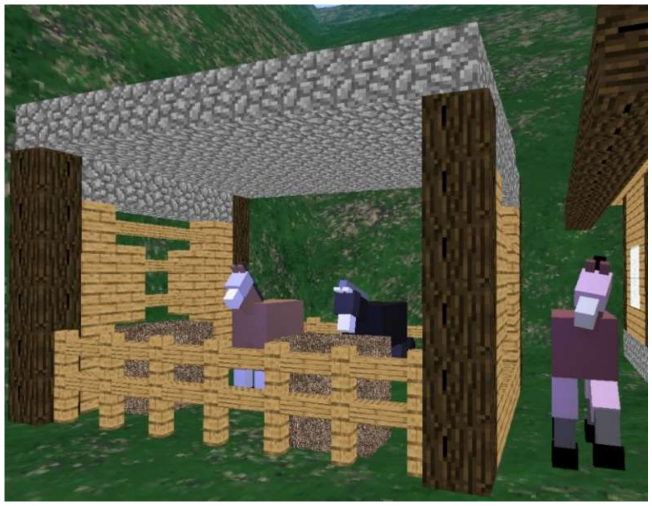


Figure 2.2: The stable with horse model.

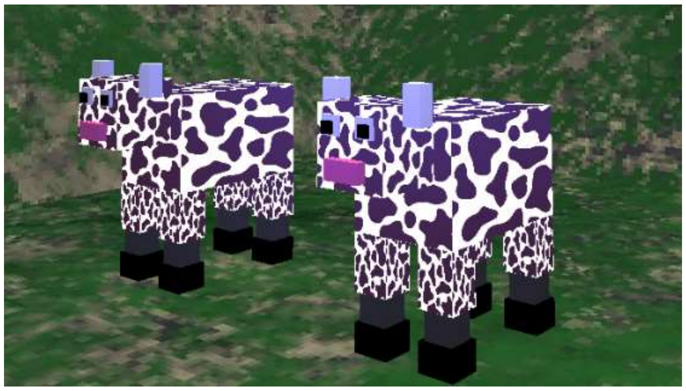


Figure 2.3: The cow model.

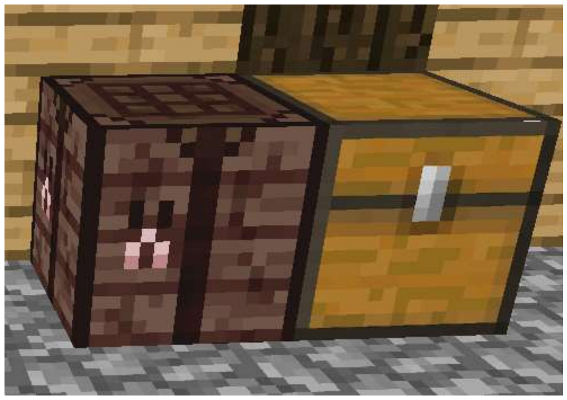


Figure 2.4: Cubes with different texture on each side.

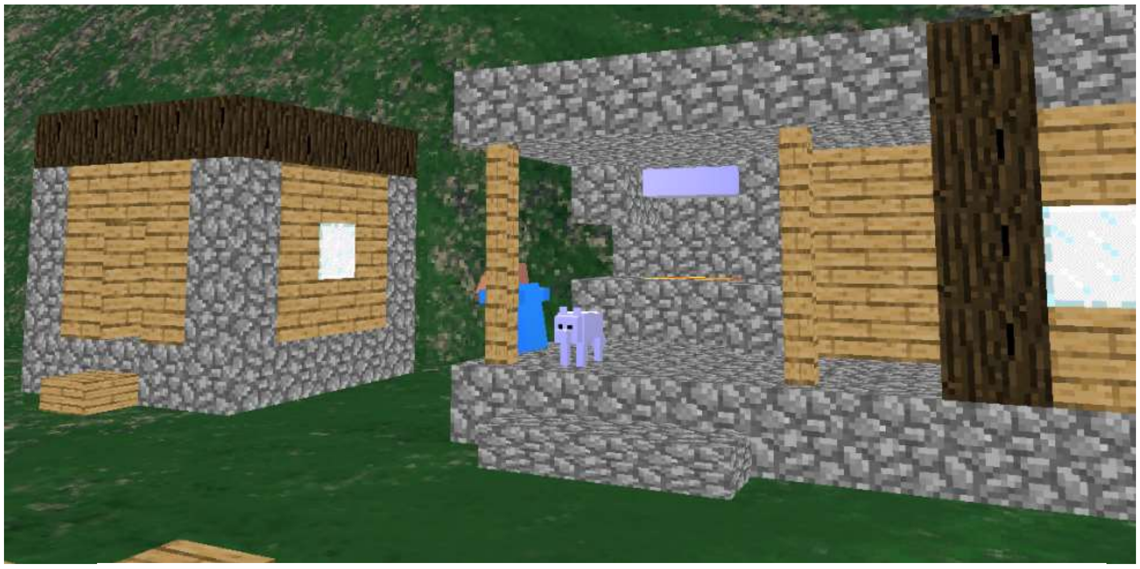


Figure 2.5: The cabin (left) and blacksmith (right) with a dog and human inside model.

The models in the scene are built through the combination of the different types of cuboids with the use of hierarchal modelling. Each model consists of several stacks that defines the different parts of their body or structure. After building the models with hierarchal modelling, they are each turned into their own function so that it can be used to populate the scene. The functions that draw the buildings are stored in the drawBuildings.h, header file and the functions for stationary animal models are stored in the drawStaticAnimals.h, header file. By turning the models into its own function, allows it to be placed in the scene robustly and saved for future use. The model that is made with the subdivision technique is placed at the top right corner of the Sky Box, which represents the sun in the scene.

3.0 <u>The Animations</u>

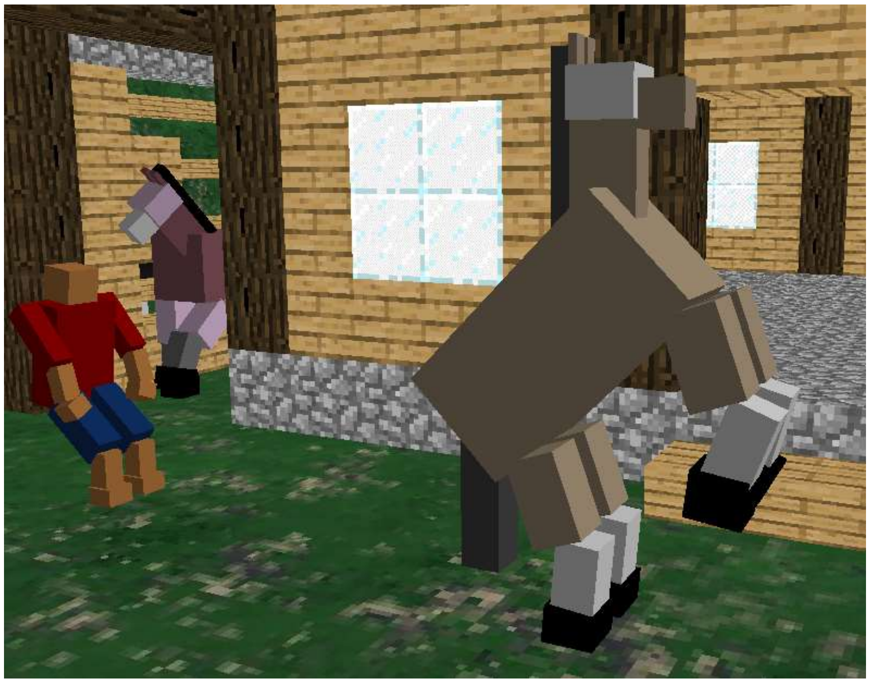


Figure 3.0: Horse rearing while the man is panting animation.

The animations in the scene are made of the models that are already made through hierarchal modelling but with added keyframes to move their joints for animation. Each keyframe consists of specific angles that instructs the movement of the model that creates the animation of the model. Each animation is also turned into their own function and stored into the, drawAnimatedAnimals.h, header file, so that it can be used to populate the scene easily as well as saved for future uses. A total of 6 different animations can be found in the scene which portrays different events happening. The first 2 animations that can be seen is a man chasing a runaway horse where the animation have 24 number of animation stages each. To allow the animation to look more realistic both the human and horse have a corresponding angle variable for each of their joints from each leg or hands. Additionally, the body and the head also have their own respective angle to animate a more realistic scene. Throughout the animation, the man is seen chasing the horse from behind. Between the 5th keyframe and the 17th keyframe of the animation both the horse and man will stop and perform their individual animation. The animation of the horse will be rearing, where it stands up on its hindlegs and the human can be seen panting behind the horse. By combining the animations, it portrays a man chasing a runaway horse and stops to catch his breath. On the other hand, the runaway horse is running away from the man, and rears when the man is catching his breath.

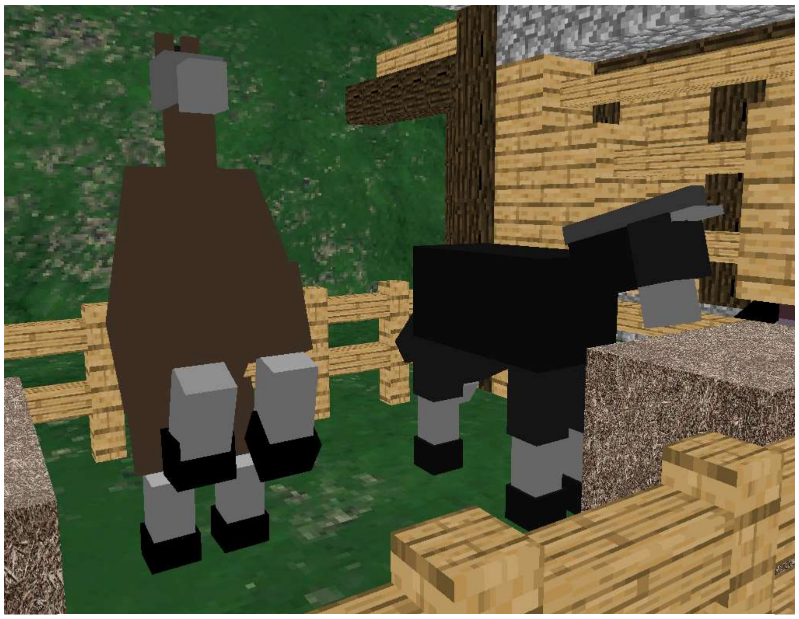


Figure 3.1: Left horse rearing while right horse is eating hay.

Another 2 animations in the scene are found in the horse stable. Both the animations consist 14 number of animation stages each. The right horse performs an animation where it turns to look to its left and goes down to eat its hay while kicking his legs to make himself feel comforable. Whereas the left horse will perform a rearing animation in the horse stable.

The final 2 animations in the scene can be found at the blacksmith's building as shown in Figure 2.5. The animations consist 14 number of animation stages each. One of the animations is a dog shaking its tail and body to show its affection. The other animation is depicting a man crying while rubbing his eyes and kicking his leg by the side of the blacksmith's building.

4.0 The Lighting & Texturing

As shown by the figures in Section 2, all the models in the scene are textured except for the horse, man, dog, and sun models. All the textured models in the scene are textured via the drawSolidCube() as stated in Section 1. An animated texture is also added into the scene at the blacksmith building. The animated texture is a textured lava cuboid that changes its texture through animation. The textures of each side of the cuboid are changed by mapping each side to different parts of the original texture and changes the part over time. This allows the lava to look like it is moving.

There are a total of 4 lights initialized in the scene. The first initialized light, GL_LIGHT0, is a directional light that is used to light up the scene. The second light, GL_LIGHT1, is a spotlight that is attached to the camera where the user can use to shine on models in the scene. The third light, GL_LIGHT2, is a positional light that is positioned above the lava texture. The light direction of GL_LIGHT2 is set to 1.0f in the Y-Axis, which allows the lava to look like it is emitting light, providing it a realistic look. The final light in the scene, GL_LIGHT3 is another positional light that is set beside the sun and directed towards the sun. Due to the reflective triangles used to create the sun, it allows the light shined on the sun to be reflected on the surface, providing the sun a realistic look that resembles a shining sun.

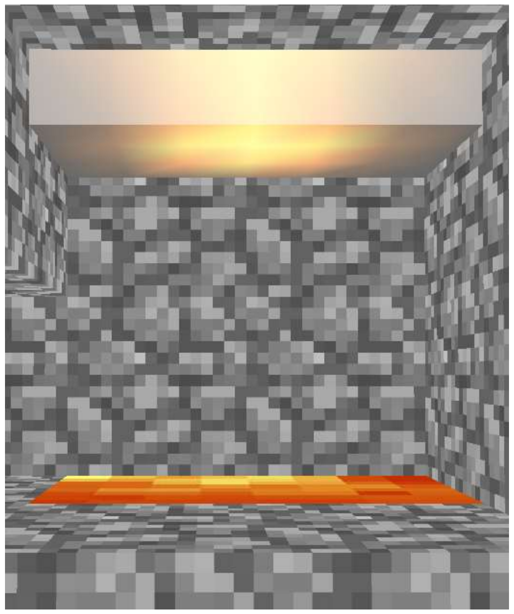


Figure 4.0: Textured animated lava which emits light in blacksmith building.

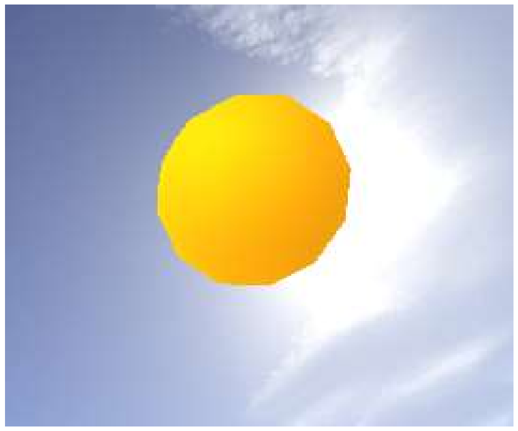


Figure 4.1: The shining sun.

5.0 The Interactions

The keyboard interactions in the scene are done through the key_callback() saved in the checkKeys.h, header file. The first distinguishable interaction available is the free movement of the camera in the scene with the keys, 'W', 'A', 'S', 'D', where it allows the user to move freely around the scene to see the different parts of the scene. By pressing 'Q', the user can switch the spotlight, GL_LIGHT1, on or off. This gives the user a feel of having a torchlight that can be used to shine on different parts of the scene.

The second interaction that can be found in the scene is a localized trigger on the dog's animation. If the camera is within the defined boundary, the user can press the key 'E' the start the animation. This reinforces the rule that the user must close enough to the dog to pet it by pressing the key 'E', that in return triggers its animation. This is done through checking the position of the moving camera in the scene when the key 'E' is pressed. If the moving camera is close enough, it will flip the Boolean, isDogAnim, to true which starts the dog's animation. Once the animation ends, the isDogAnim is flipped back to false.

The third interaction is another localized trigger on the rearing horse's animation in the horse stable. To start the animation the user needs to press the key 'E' within a specified region from the horse. Once the 2 conditions are met, the horse will start its rearing animation, depicting an event where the user is annoying the horse which resulted in the horse rearing. This is also done by checking the camera's position when the key 'E' is pressed. Similarly to the dog's animation, the horse's animation has its Boolean, isHorseAnim, set to true that starts the animation when all the conditions are met and returns to false when the animation ends.

The fourth and final interaction is the closeness of the camera to the sun. As the user gets closer to the sun, the subdivision depth of the sun increases. When the scene is first started, the subdivision depth is initialized to 2. As the player moves towards the sun, the subdivision depth will increase accordingly based on the camera's position. There are three subsections defined that increases the subdivision depth of the sun. The first subsection increases the sun's subdivision depth to 3, the following subsection increases the subdivision depth to 4, and the last subsection increases the subdivision depth to 5.