DH2323 LAB 3

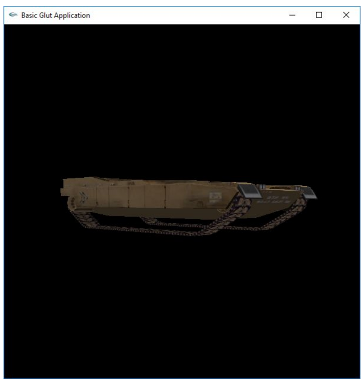
1 Display and Animate the Tank Hierarchy

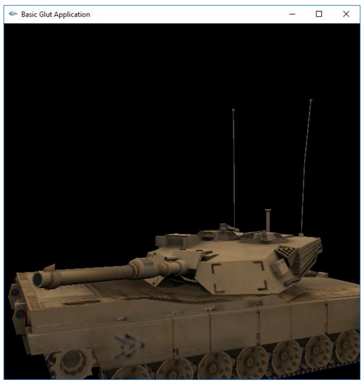
1.2 Tasks

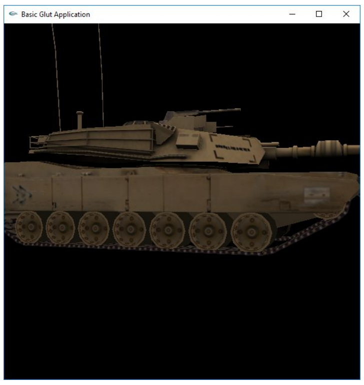
Task 1: Assemble the rest of the tank.

This was simply done by translating the different parts of the tank until it looked good. The matrixes for each part was pushed and then popped while adjusted. They all depend on the TankBody and the MainGun and SecondaryGun also depends on the Turrets position as well.

Several camera keys were also implemented to make it easier to see when a part fit correctly. Both xPos, yPos, zPos and rotation is available by using the keys a,d; w,s; q,e; r,f respectively.



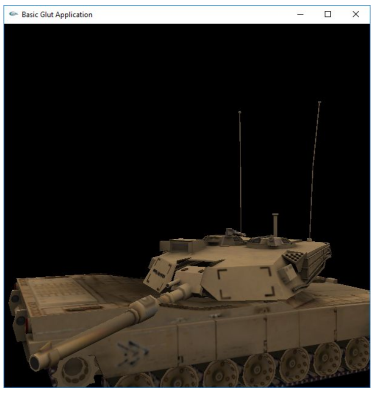


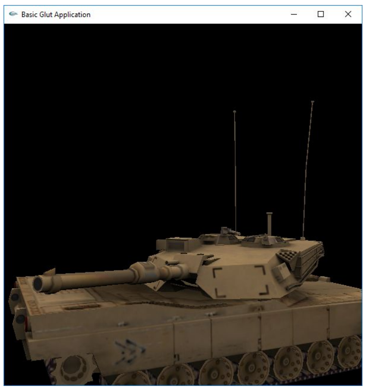


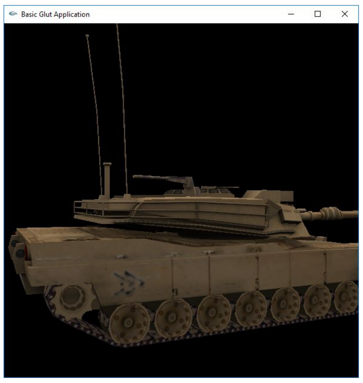
The wheels were implemented with a for-loop to avoid unnecessary redundant coding.

Task 2: Allow the user to rotate the turret, main gun and secondary guns, and wheels on the tank by pressing various keys.

Since the hierarchy of the parts were already in place this turned out to be really simple by just adding a key for each action in the function key() and then rotate the parts accordingly.







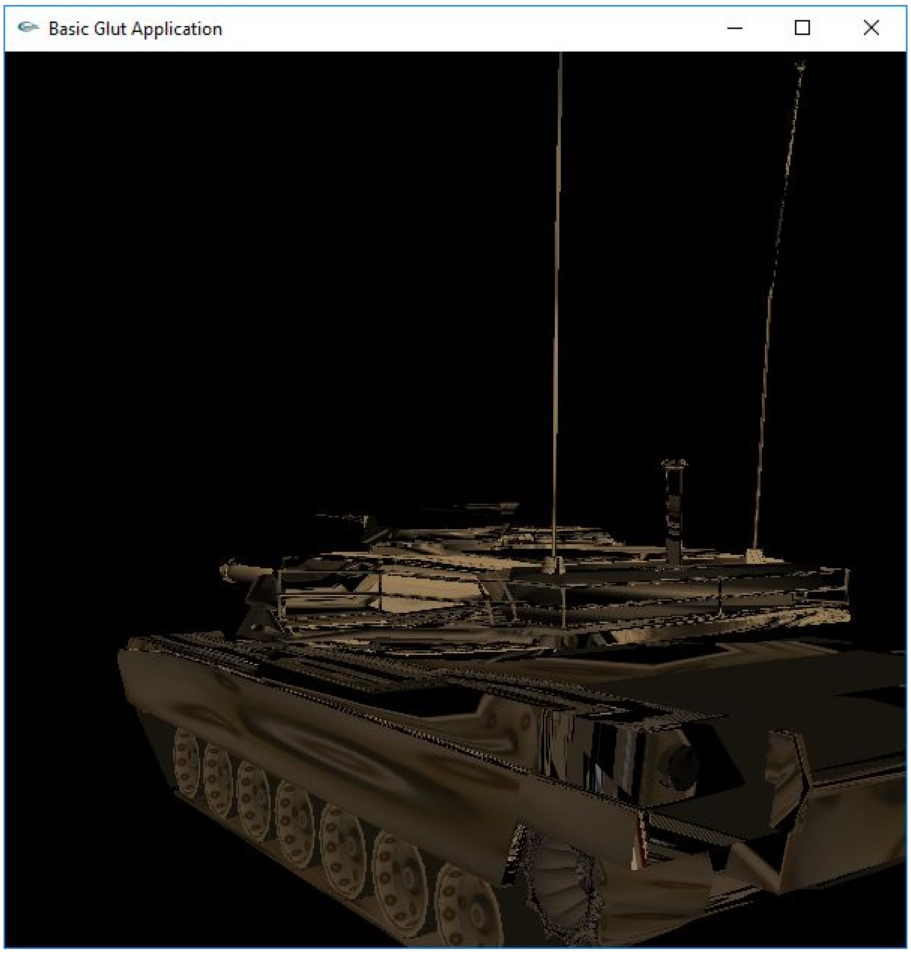
The Turret can be rotated with key 1&2. The MainGun can move up and down 5 pixels in each way using key 3&4. The SecondaryGun can be rotated using key 5&6. Lastly the wheels can be rotated with key 7&8.

Task 3: Add each component of the tank into its own display list.

This was done by following the instructions in the lab spec. First each part got allocated a space in the GenList. Then a new list was created and each object loaded into respectively list. Then callList was used to call the objects when they were to be drawn into the canvas.

Task 4: Implement your own DrawObj(. . .) function.

This was a bit of a challenge. By implementing the pseudo code in the lab spec I almost managed to get it right. What I missed was that the k values for the vertices, normals and textures array were different, so my first implementation ended up in the image seen below.



2 Bounding Volume Hierarchy (BVH)

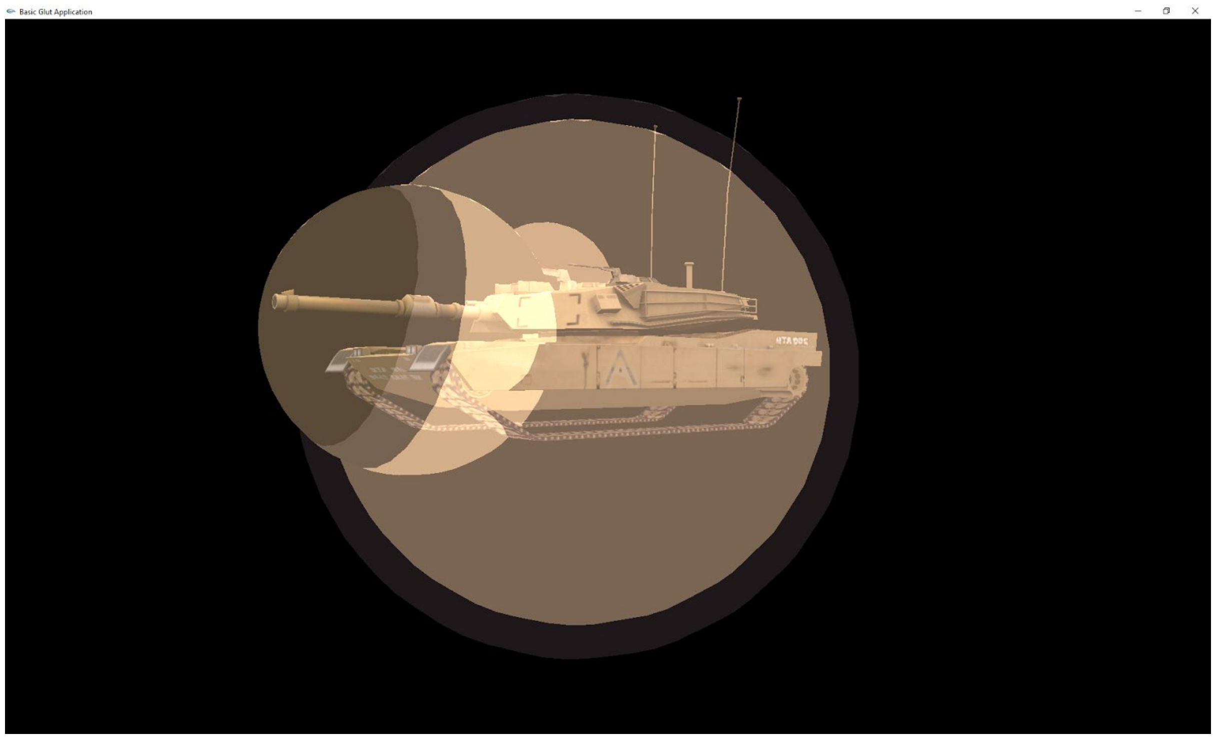
2.2 Tasks

Task 1: Create a new BoundingSphere class which stores the bounding sphere details for a single mesh.

This seemed like an easy task from the start, but I had great problems finding the average vertex point for each mesh. Thus leading to difficulties finding the radian of the boundingSphere. This was due to dividing directly with the mesh-faces. Still don't understand why, but it was fixed by assigning its value to a variable and dividing by that variable instead.

Task 2: create and store a bounding sphere for each of the sub-objects and for the whole tank object.

This was done quite easily by translating each of the spheres until they surrounded their respective subPart. However translating by trial and error takes some time, but I couldn't figure out if there were any other way to do it than that.



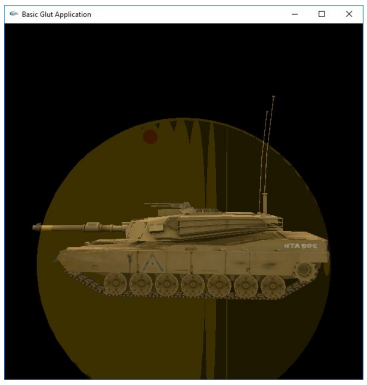
I realized at Task 4 that the wheels weren't showing and simply let them be created before the turret which fixed the problem. It's shown in later images.

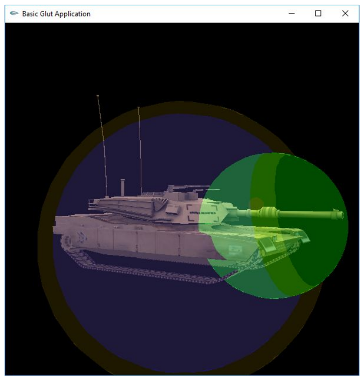
Task 3: Create a function to test if a given (x, y, z) position is potentially penetrating (a) the tank bounding sphere and, if so, (b) its subparts.

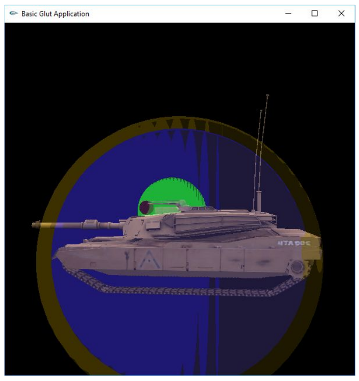
The function itself wasn't so hard to create I simply checked whether the distance between the two points were shorter than the boundingSpheres radius and if so they collided. However due to the translation being done earlier I had to hardcode this part somewhat to make the collision act as it should.

Task 4: Extend the previous question to display the point (representing a projectile) on the screen and allow the user to move it around interactively using the keyboard.

This part was a nightmare to implement. This was due to the transparency of the boundingSpheres and the projectile itself. It was merely impossible to see the projectile and I struggled with this problem for hours until I realized I had to reset the coloring after each boundingSphere creation. The main problem was that I didn't realize the texture of the tankwent back to normal when setting glcolorf to (1.0,1.0,1.0). After I realized this I colored the rest of the spheres to different colors depending on their hierarchy level. Yellow for the body, blue for the turret and green for the Guns. When the projectile intersects with a boundingSphere, the boundingSphere becomes visible.

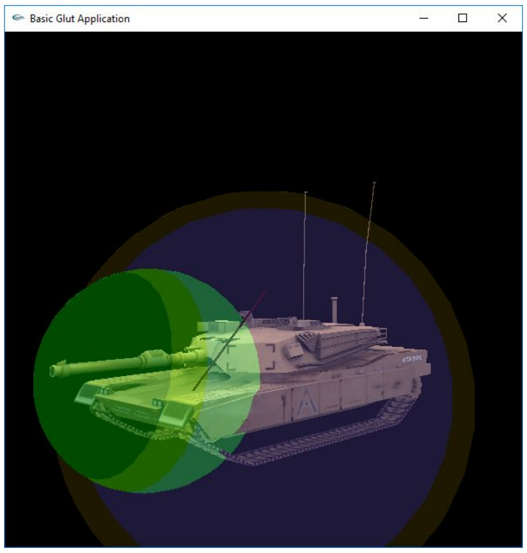






Task 5: Create a function to test a line, rather than a position, with the bounding sphere hierarchy.

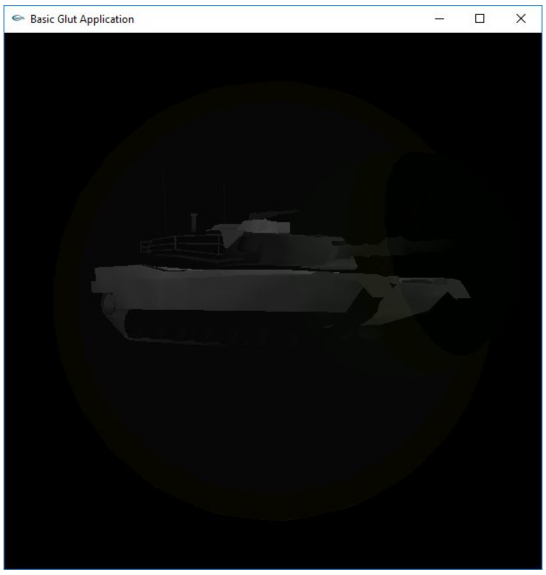
This was done by checking the closest point on the line to the origin of each boundingSphere. When the point had been found the same approach as in the projectile's intersection was used.

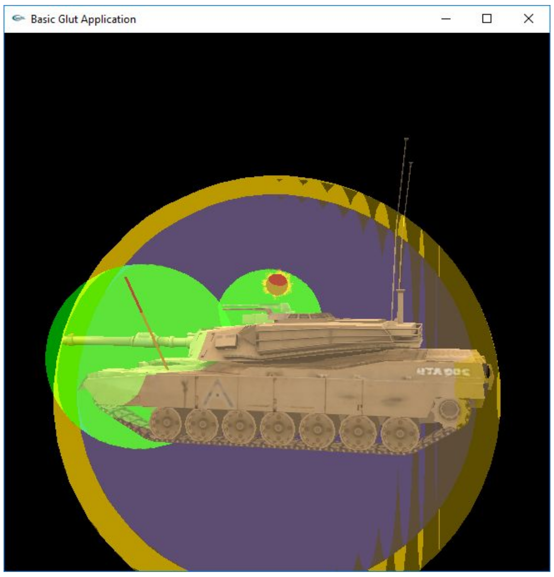


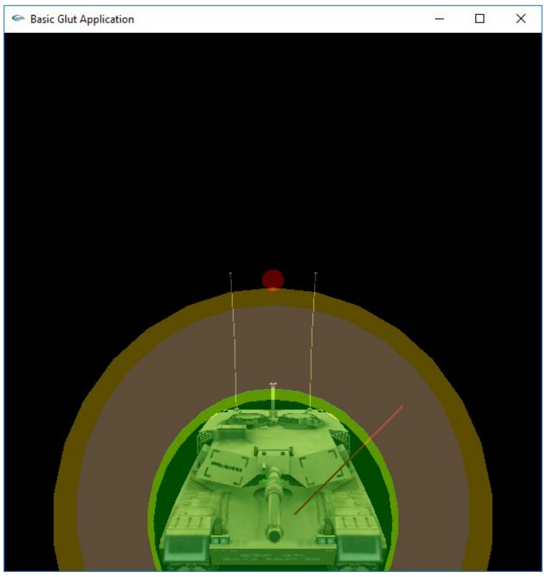
The code need to be uncommented in the draw_tank function to show the line.

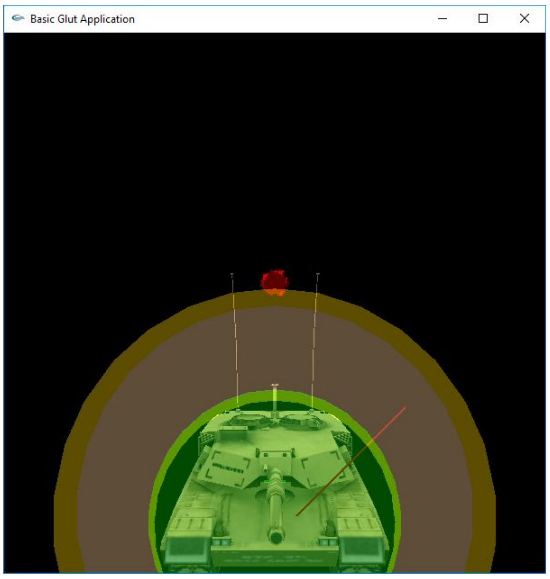
Task 6: Integrate the particle system code from Lab 2. Generate sparks when the projectile is potentially colliding with the tank centered around the position of the projectile.

This took quite some time to integrate. This was mainly due to many functions being ambiguous and was solved by using ifndef guards around all .h-files. It was a bit tricky to get particles into place and the absolute biggest problem was the texturing. I still haven't managed to figure out how to reset the texture after it being set by the bindTexture. The result is shown in the upper left picture below. I was able to use the setTexture after each draw to make the tank look normal, but this operation is really slow resulting in the application lagging when moving the projectile. In the end it was easier to remove that little part of texture since it still looks good and the application moves smoothly.









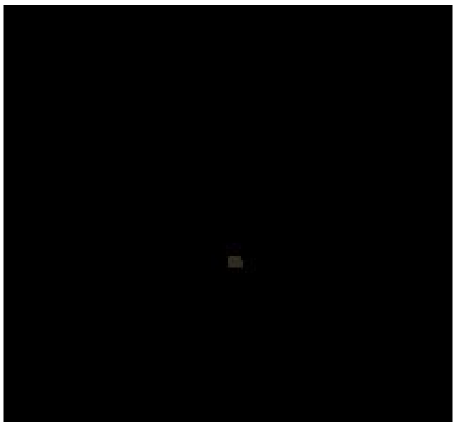
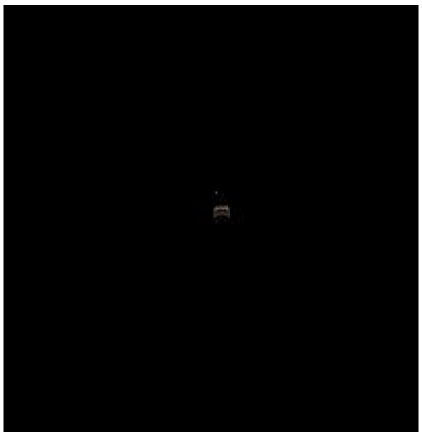
3 Discrete Level Of Detail

3.2 Tasks

Task 1: Define a new low detail level version of the tank.
This was defined by the same way that the high-level-detail version was implemented and accordingly to the lab spec.

Task 2: Find the threshold distance (d) when there's no apparent difference between the high- and low-level detailed tank.
Firstly I had to change the gluperspective's zFar to 1000 otherwise the tanks were no longer visible after 100px. By measuring the zPos and putting the two versions into the same image I moved the two tanks backwards until I saw no difference. This was when zPos was around -650 and therefore I put d to -650.

Task 3: Implement a discrete LOD technique based on viewing distance so that, as the viewer presses a key to move the tank further away from the object, a level of detail switch takes place beyond distance d. above.
This was simply done by using an if statement if zPos > -650 draw_high_detail_tank else draw_low_detail_tank. The low-detail tank is shown to the left below and the high-detail to the right.

In summary this lab was definitely the hardest, but I still learned a lot about graphics and C++. What is annoying however is that the setup of the labs are so time-consuming and you spend so much time trying to get the code to compile. I understand that this is how it works in the real world and it's good to get the experience but it would be so much more fun to just focus on the coding and functionality rather than trying to integrate different parts of code into other projects.