Unit Material
- Home
- Handbook Entry
- Unit Timetable
- Lecture Slides
- Lecture recordings
- Laboratory Sheets & Project

Unit portals

Useful Resources
Recommended Text:
- Web site
- WebGL - run in a browser

OpenGL:
- OpenGL Documentation
- OpenGL4.5 and GLSL Reference Pages
- OpenGL Wiki
- OpenGL FAQ
- Programming Book (free)
- Common Mistakes

freeglut/GLUT
- GLUT API
- freeglut API

C/C++:
- C and C++ for Java Programmers
- C++ and Java Syntax Differences Cheat Sheet
- C and C++ For Java Programmers (actually just C)

Department of Computer Science and Software Engineering

CITS3003 Graphics & Animation 2021 — Project-2021/project-2021

UNIT COORDINATOR & LECTURER

Dr. Naveed Akhtar

LAB FACILITATORS

David Charkey
Matthew Chidlow

CONSULTATION TIME

Wednesdays, 4:00-5:00pm.
Where: Room 1.12 in CSSE and online (active in consultation hour)

NEWS:
- [22 Feb'21] Welcome to CITS3003
- [30 March 21] Midsemester test 10:00AM. Use this Zoom link to join the Zoom meeting during the test.
- [13 Apr'21] Programming project released
- [15 Apr'21] Project template updated for Mac build. If needed, see the changes from last version here

Project submission due on Tuesday 18.05.2021 (1700 hrs)

The project will be marked out of 100 but is worth 40% of the total unit marks.

Groups: Project needs to be done in groups of two. However, you can also choose to do it individually. There is no extra credit for an individual submission though. You are free to choose your project partner. One member of each team needs to inform us about the project partners by Friday 07.05.21, 1700 hrs (end of week 10). You need to provide this information by emailing to naveed.akhtar@uwa.edu.au, as follows:

Subject of email: Project team CITS3003
Email content:
- Group member 1 student ID and full name, group member 2 student ID and full name.
- Answer to 'Does your group want an online (i.e. Zoom) demo, or an in-person demo in the lab?'

If no email is received by Friday 07.05.21, 1700 hrs as your name/ID in any group, we will assume that you are doing the project individually. You will be expected to demonstrate your project in-person individually in that case.

The above instructions apply to both online and face-to-face students, where online students 'must' demonstrate online.

Submission: You will submit your files through LMS. The submission link will appear in Week 10.

- You must submit all your files including header files and base code (do not submit the models-textures, we already have those.). To ensure that your program works when we test it, **you should submit a zip file that retains the directory structure of the program. Note:** Please provide a ReadMe file with your submission that clearly states the OS used for the project. You can also specify any other requriements or procedure to run your project.
- For the report mentioned below, it should be in **PDF** format. Please mention your full names and student numbers on the first page of the report. Please name your report file as **report_student1ID_student2ID.pdf**.
- Each group will finally have to demonstrate the project. You can either demonstrate on your machine or any lab machine. You must have submitted your project before the demonstration. The demo is the chance for you to show what works in your project. Expect cross-questioning during demonstrations. Your marks for each part will also depend on your answers to the questions.

Originality: You may discuss with other students the general principles required to understand this project, but the work you submit by a group must be the sole effort of the group members. You are allowed to base parts of your code on the lab sample solutions, as well as the examples from lectures and the recommended text. If you base your code on code from any other source you must clearly reference the origin in a comment. Your comment should also mention exactly which parts of your code are based on the code from that origin. It will be considered academic dishonesty if you omit any such references.

Detailed specification for the project are given below.

Project

The main project task

You are required to complete an implementation of a simple scene editor that allows a collection of objects to be arranged in a scene and various properties of them to be changed, such as colour, shininess and texture. The specific items you have to implement are below, including sample videos.

Files provided

1. Starting point - download the **project-2021.zip** file and extract somewhere safe. Inspect the skeleton file **scene-start.cpp** in the **src** subdirectory. Regardless of your OS, this is the main file you should be able to edit to complete the project. You will also need to edit the shader files in the **res/shaders** directory. Study the **README** file. We will also keep a **live link** to this file during the project phase to incorporate any student feedback to the instructions. The project requires you to use the mouse to move objects and light sources around. Click here if you need help to set up a 3 button mouse on the Mac.
2. You will also have a zip file that contains many texture map files. This file can be downloaded **here**. Note that, the provided files are for educational purpose only and **cannot be publicly distributed**. The zip file contains a subdirectory named **models-textures** which should be placed in the **res** directory.
3. The project will build all the dependencies you need from source (except for when using the lab pc's or a mac, there it will use precompiled/OS provided libs for some or all as needed)

Project report

With your source code, each group must submit a project report that describes:

- For which parts of the project your solution works fine? Also mention the part(s) that you were unable to do.
- The steps you followed in building your program, focusing on the problems that you needed to solve in this process. This description can be brief.
- For those who went beyond what is asked for in the project, you can use this document to also report the additional functionalitise.

There is no specific format of the report, and there are no explicit marks for this document. However, it has a very specific purpose. At your demo, you should expect your marker to have seen your report beforehand. Your report should be able to clearly show what worked in your solution (which you can later demonstrate). In your report, you can also take help from figures (e.g. screen shots) if you like. Mention separately, what tasks you are unable to do. Do not use the report to give OS specific details. Use a ReadMe file for that prupose, which can be as detailed as you like. Note that, the report is a necessary document for project submission. You may get heavily penalized for not submitting it, because your marker may not be able to assess your project accurately during a few minuts demo. Use this document wisely to secure the best grades and make your demo smooth. Only one report and one submission is required from each group.

Assessment Details

Your project has 100 marks and it will be assessed based on correctness of the functionalities and coding (structure, clarity and appropriate use of OpenGL), and your understanding reflected during the demonstration. Each task carries 10 marks. Partial marks are possible with partially correct solutions.

Specific tasks that your scene editor should implement

Your scene editor needs to implement all the functionalities described in the items **A** to **J** below.

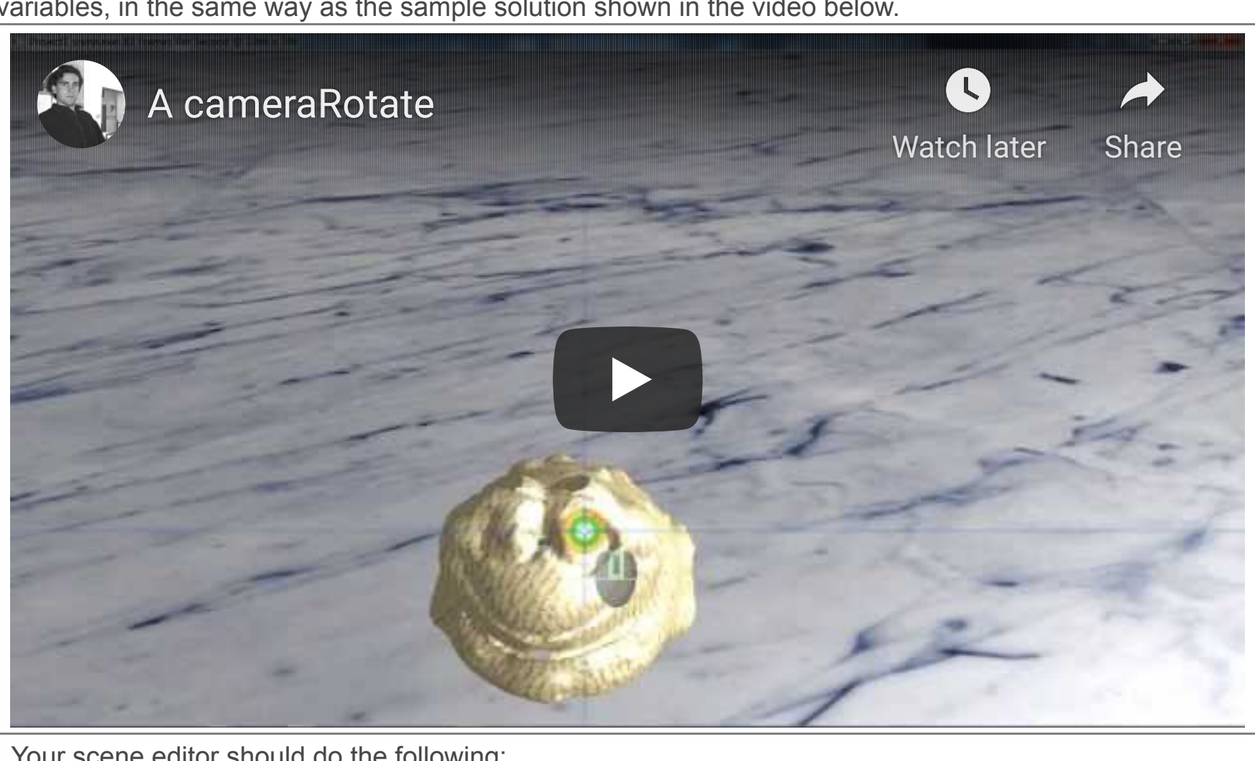
Don't panic if the videos provided alongside the items seem beyond what you have done in the labs. The project tasks have been chosen to cover all the relevant concepts without requiring you to implement too much extra. Thus, much of the functionalities are already implemented in the skeleton code, and some of the tasks require similar code to what is in the skeleton code.

Also, you are allowed to improve upon the specification and implement some items differently, but you should explain in your report why it is an improvement.

Although the videos provided below don't have any audio, all the mouse actions were captured in the videos. We suggest that you view each video in full screen mode carefully. Please pay attention to the mouse icon shown inside the window.
- When the circle next to the mouse icon turns into red, it indicates that the left mouse button is held down.
- When the circle becomes green, it indicates that the middle mouse button is held down. When the wheel of the middle mouse button is rolled, you should see an arrow and its direction.
- The right mouse button is reserved for bringing up the menu.

For your convenience, you can download the Zip file [87MB] of all the videos below.

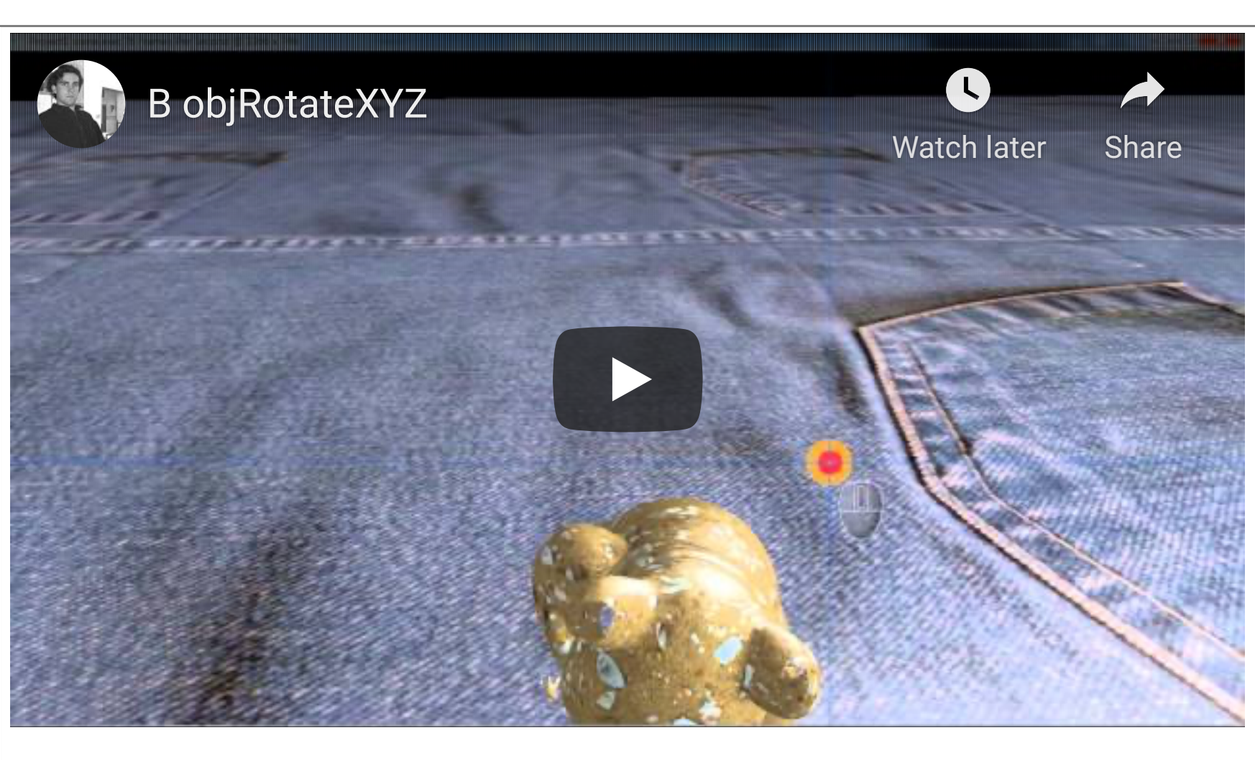
A. The skeleton code draws the ground, an initial mesh object and a sphere showing the position of a single point light source; however, it has the camera always facing the same way. You can move it forwards and backwards using the scroll wheel, via the `viewDist` variable, but there's no way to move it otherwise. You should fix this problem in the code. The variables `camRotUpAndOverDeg` and `camRotSidewaysDeg` are already set appropriately when moving the mouse with the left button down, or the middle button down (or shift + left button). Modify the display function so that the camera rotates around the centre point according to these variables, in the same way as the sample solution shown in the video below.



Your scene editor should do the following:
- When the left mouse button is held down and dragged horizontally across the window, it should rotate the scene about the axis that is vertical to the ground plane.
- When the left mouse button is held down and dragged vertically up (or down) in the window, it should zoom into (or out of) the scene.
- Dragging the middle mouse button horizontally is the same as dragging the left button horizontally.
- Dragging the middle mouse button vertically up (or down) across the window should change the elevation angle of the camera looking at the ground plane. Correspondingly, this tilts the entire scene up (or down).

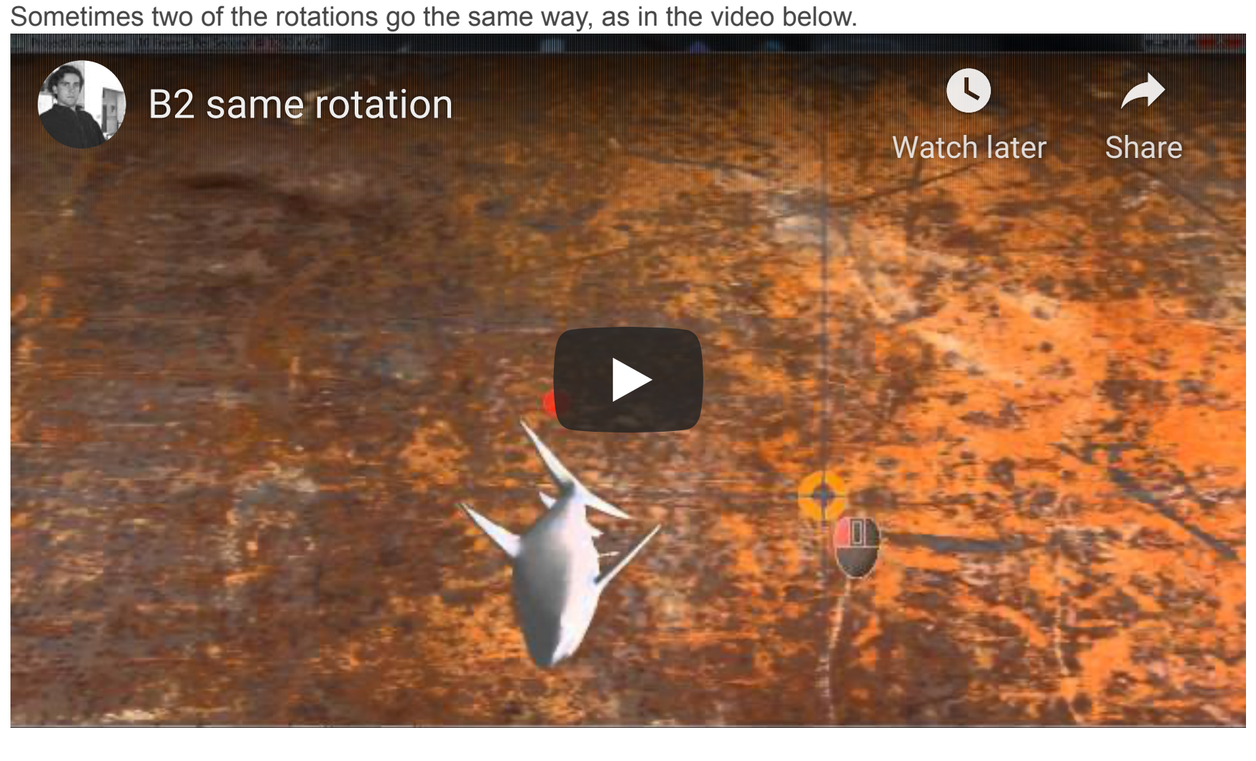
See also the link here for some additional explanation.

B. Similarly, changing the location and scale of objects is already implemented, via the `loc` and `scale` members of the `SceneObject` structure, which is stored in the variable `sceneObjs` for each object in the scene. However, the `angles` array in this structure doesn't affect what's drawn, even though it is appropriately set to contain the rotations around the X, Y and Z axes, in degrees. Modify the `drawMesh` function so that it appropriately rotates the object when drawing it in the same way as the sample solution.
Note: The skeleton code also moves objects in different directions compared to the sample solution -- you should also fix this (**Hint:** there is more than one way to do this).

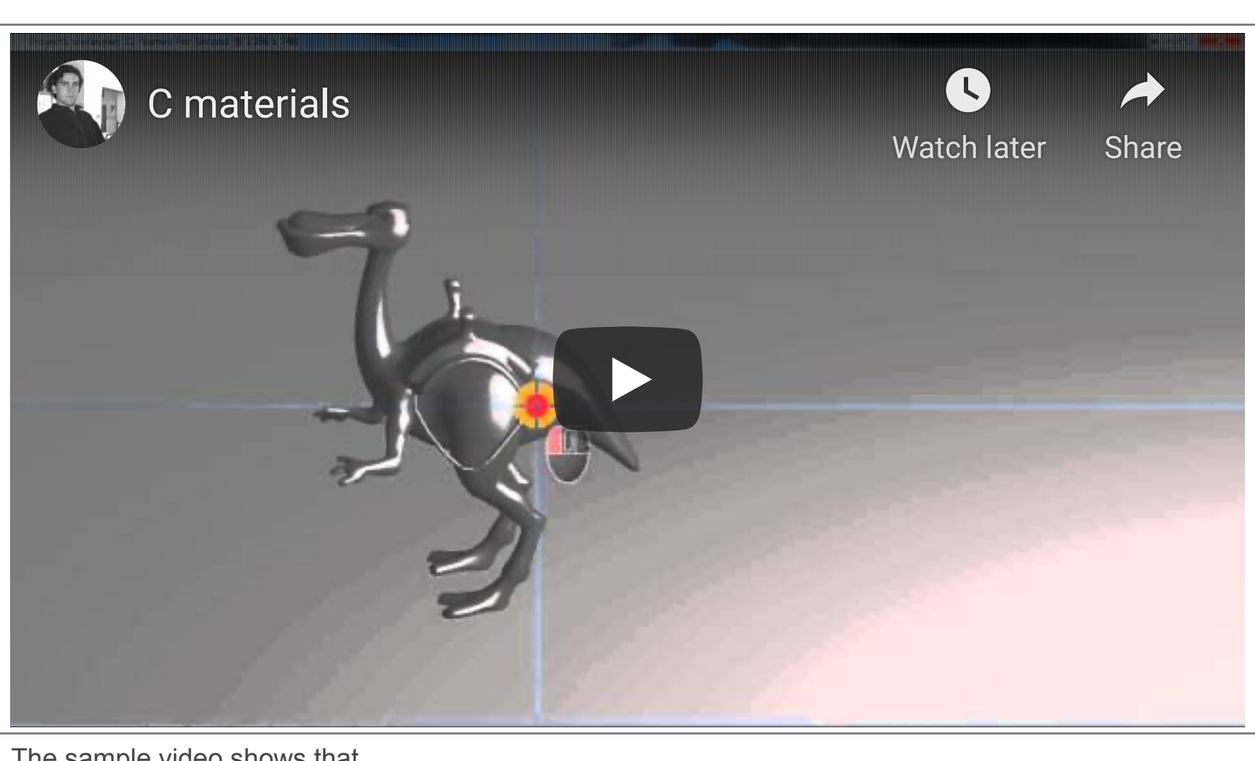


As shown in the video, your scene editor should do the following:
- Dragging the left mouse button vertically across the window should rotate the current object about the x-axis (parallel to the ground plane and pointing to the right).
- Dragging the middle mouse button horizontally across the window should rotate the current object about the z-axis (parallel to the ground plane and pointing out of the screen).
- Dragging the left mouse button horizontally across the window should rotate the current object about the y-axis (vertical to the ground plane).
- Dragging the middle mouse button vertically across the window should change the texture scale on the object.

Sometimes two of the rotations go the same way, as in the video below.



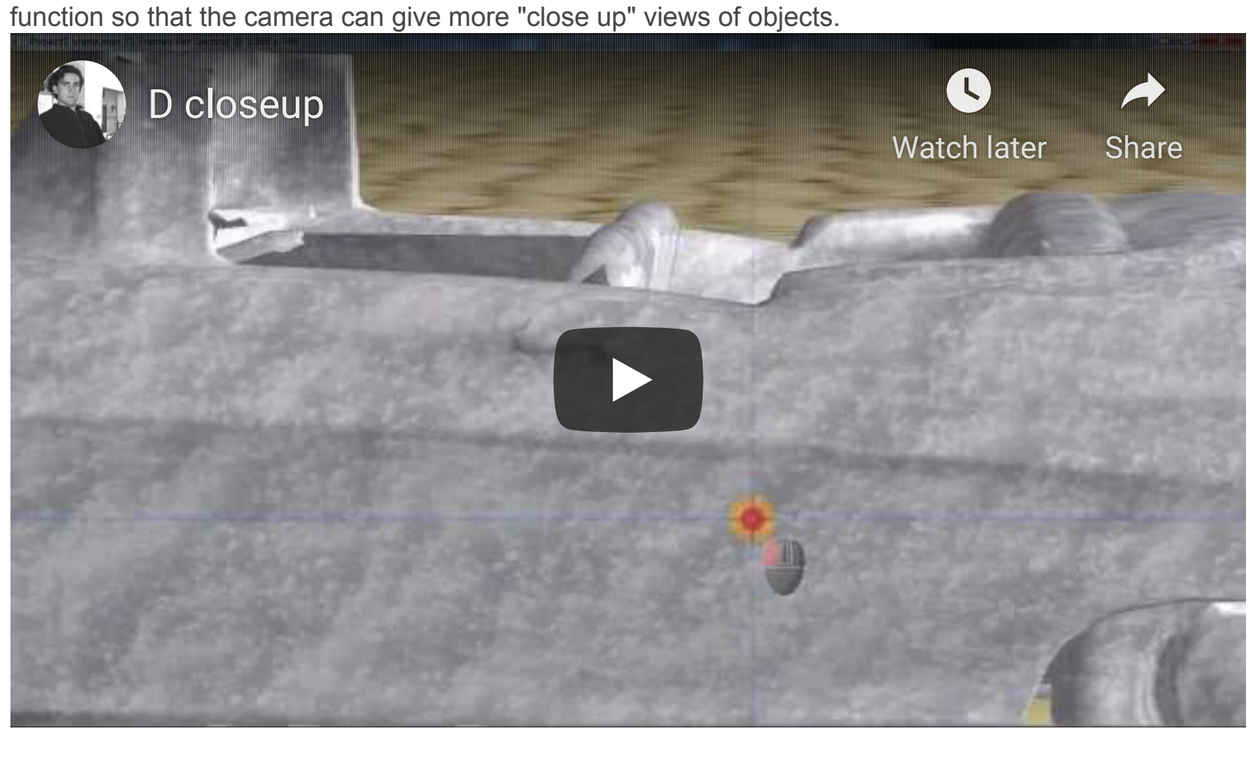
C. Implement a menu item for adjusting the amounts of ambient, diffuse and specular light, as well as the amount of shine. Recall that the shine value needs to be able to increase to about 100 or so. Modify the `materialMenu` and `makeMenuM` functions so that they change the appropriate members of the `SceneObject` structure. The code to use these to affect the shading via the calculation for the light is already implemented in the skeleton code provided.
Follow the corresponding implementations of other similar menu items. Use the `setToolCallbacks` function which has four arguments which are pointers to four float's that should be modified when moving the mouse in the x and y directions while pressing the left button or the middle button (or shift + left button). After each callback function accepting a pair of (x,y) relative movements is a 2×2 matrix which can be used to scale and rotate the effect of the mouse movement vector. See the calls to `setToolCallbacks` in `scene-start.cpp` for example. The `setToolCallbacks` function is defined in `gnatidread.h`.

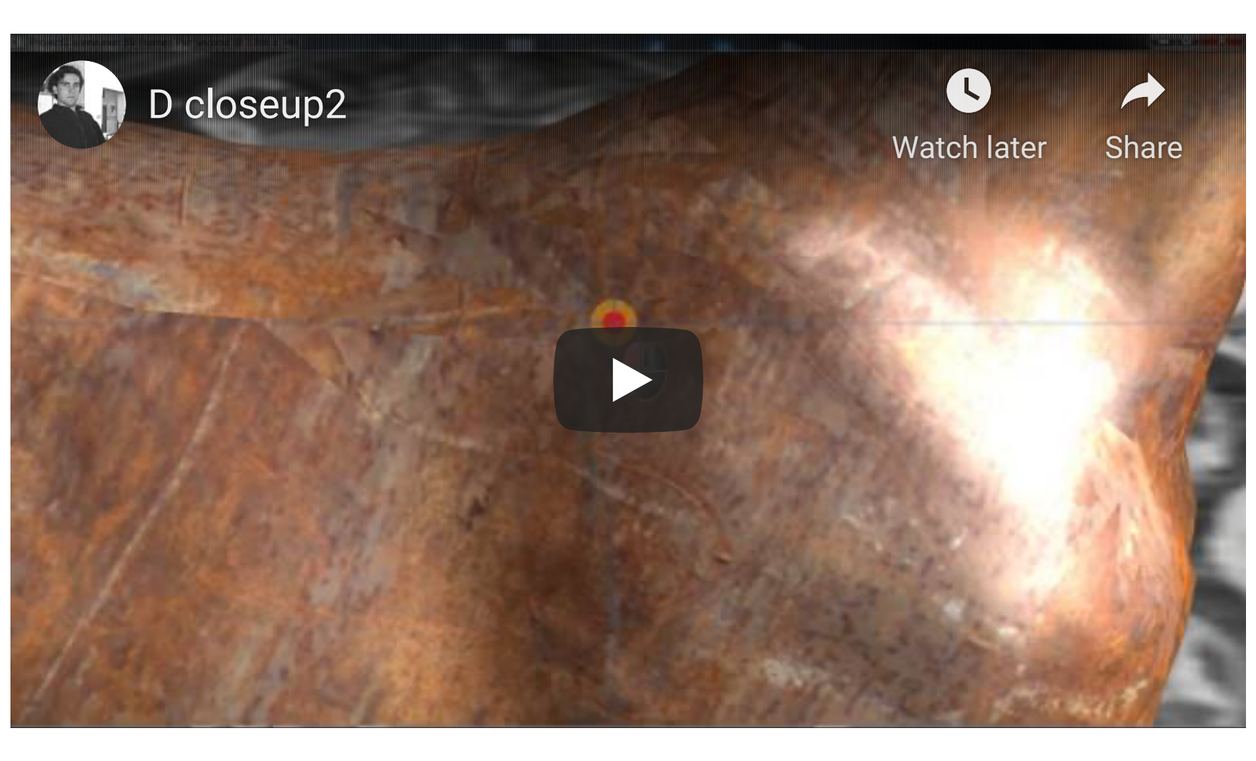


The sample video shows that
- the ambient or diffuse light of the object can be interactively changed by dragging the left mouse button horizontally or vertically;
- the specular light and amount of shine can be interactively adjusted by dragging the middle mouse button horizontally or vertically.
- the position of the light source can be modified by dragging the mouse button. For this functionality, you are suggested to use
  - the left mouse button for changing the x- and z-coordinates
  - the middle mouse button for changing the y-coordinates

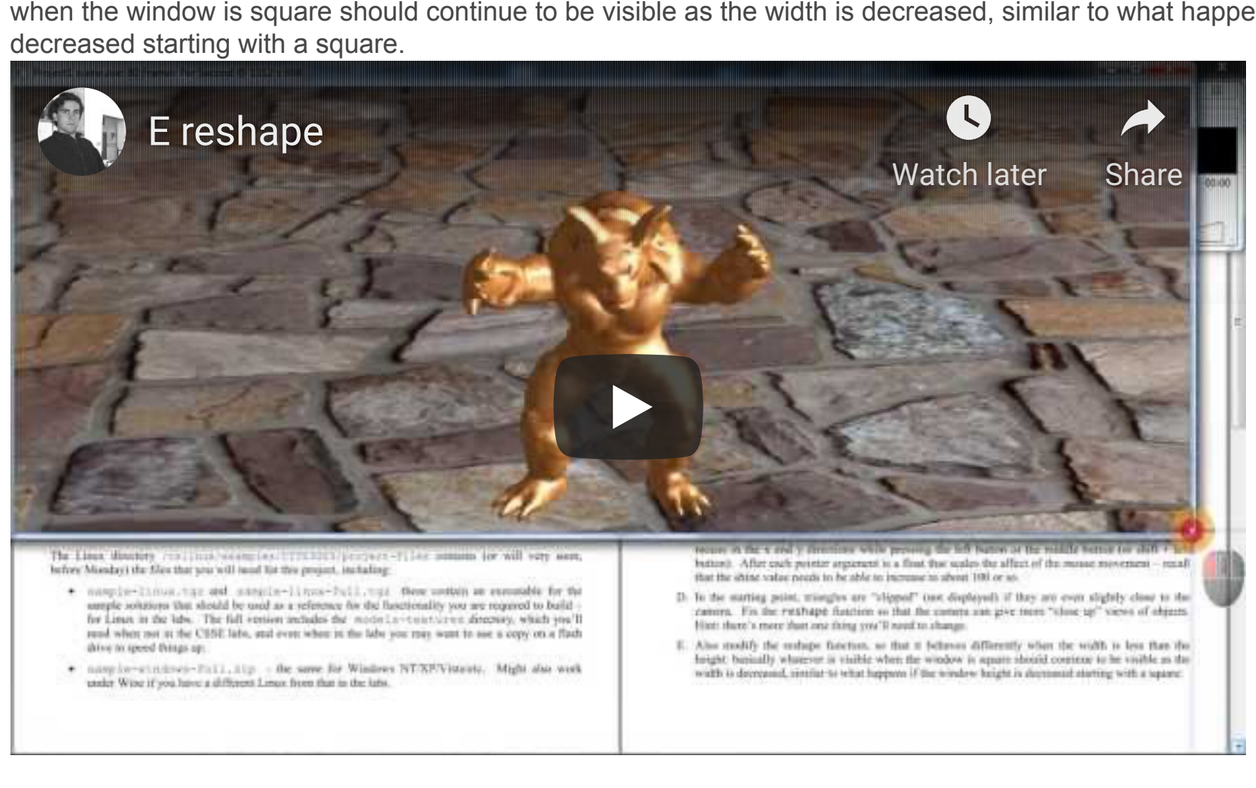
  of the light source's position

D. In the skeleton code, triangles are "clipped" (not displayed) if they are even slightly close to the camera. Fix the reshape callback function so that the camera can give more "close up" views of objects.

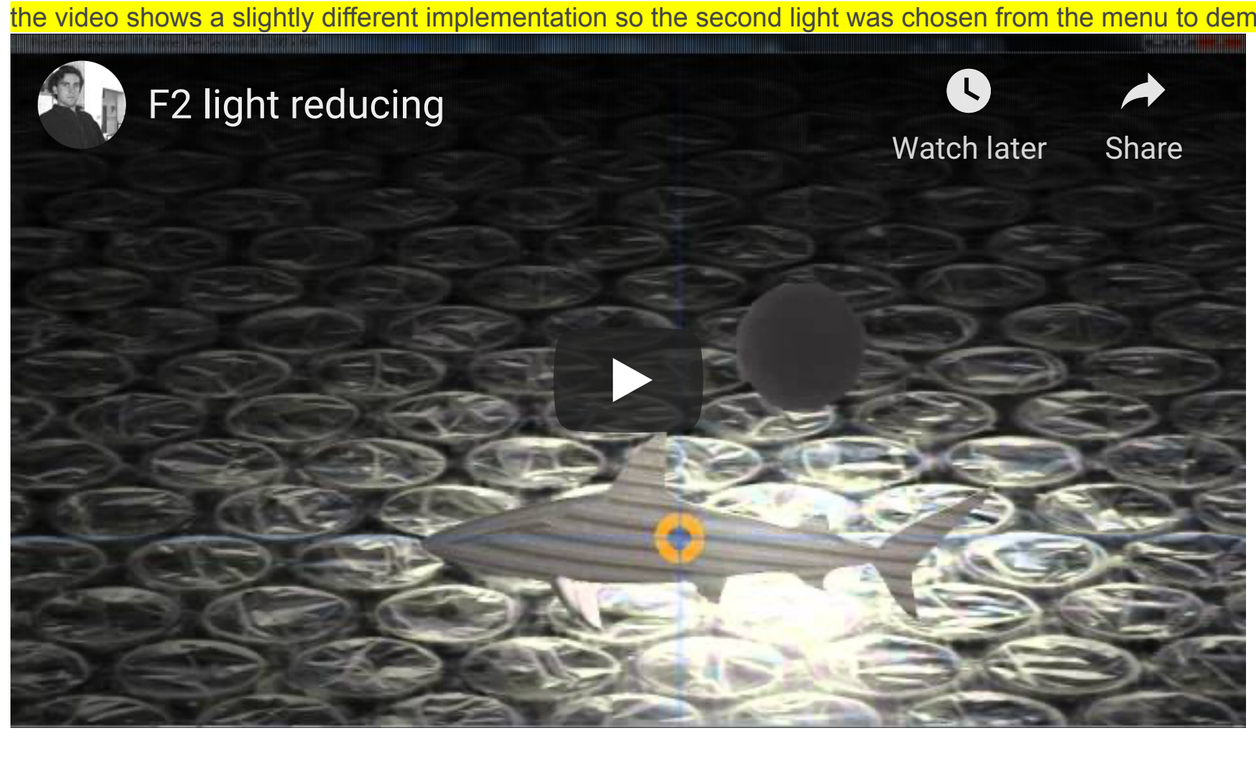




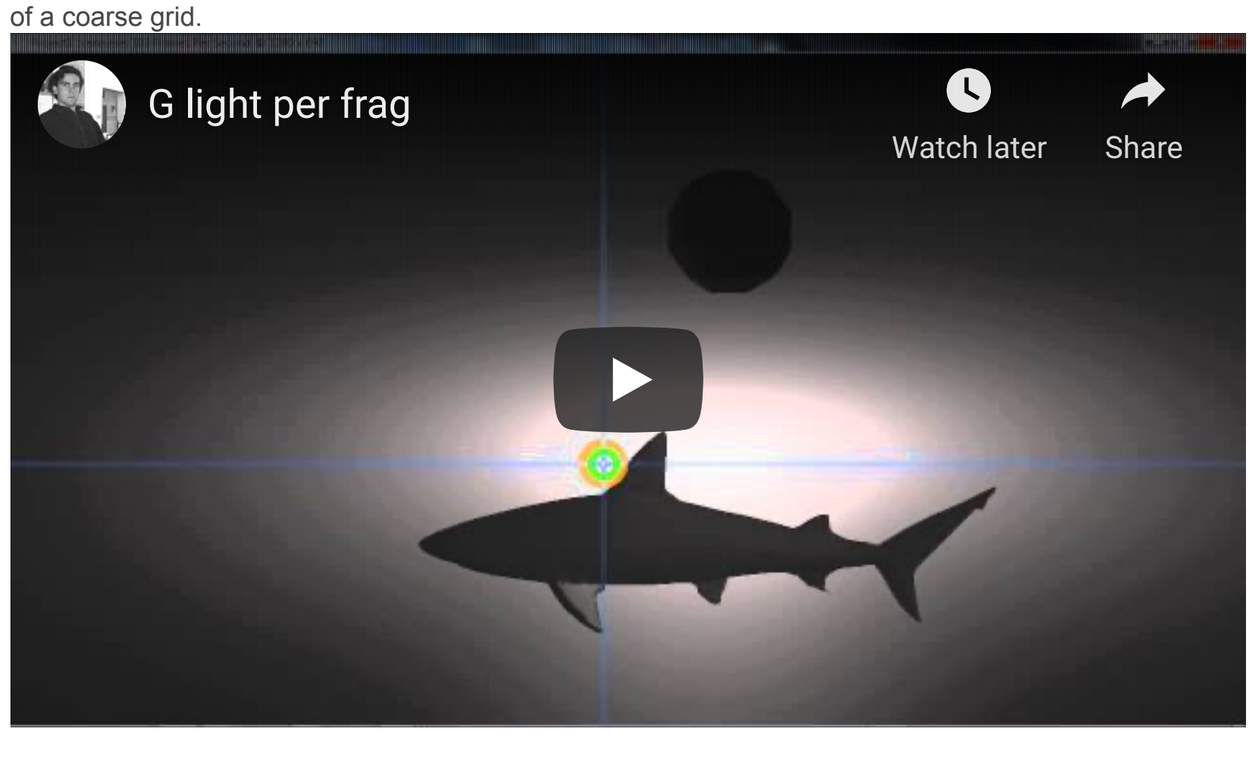
E. Also modify the reshape function, so that it behaves differently when the width is less than the height: basically whatever is visible when the window is square should continue to be visible as the width is decreased, similar to what happens if the window height is decreased starting with a square.



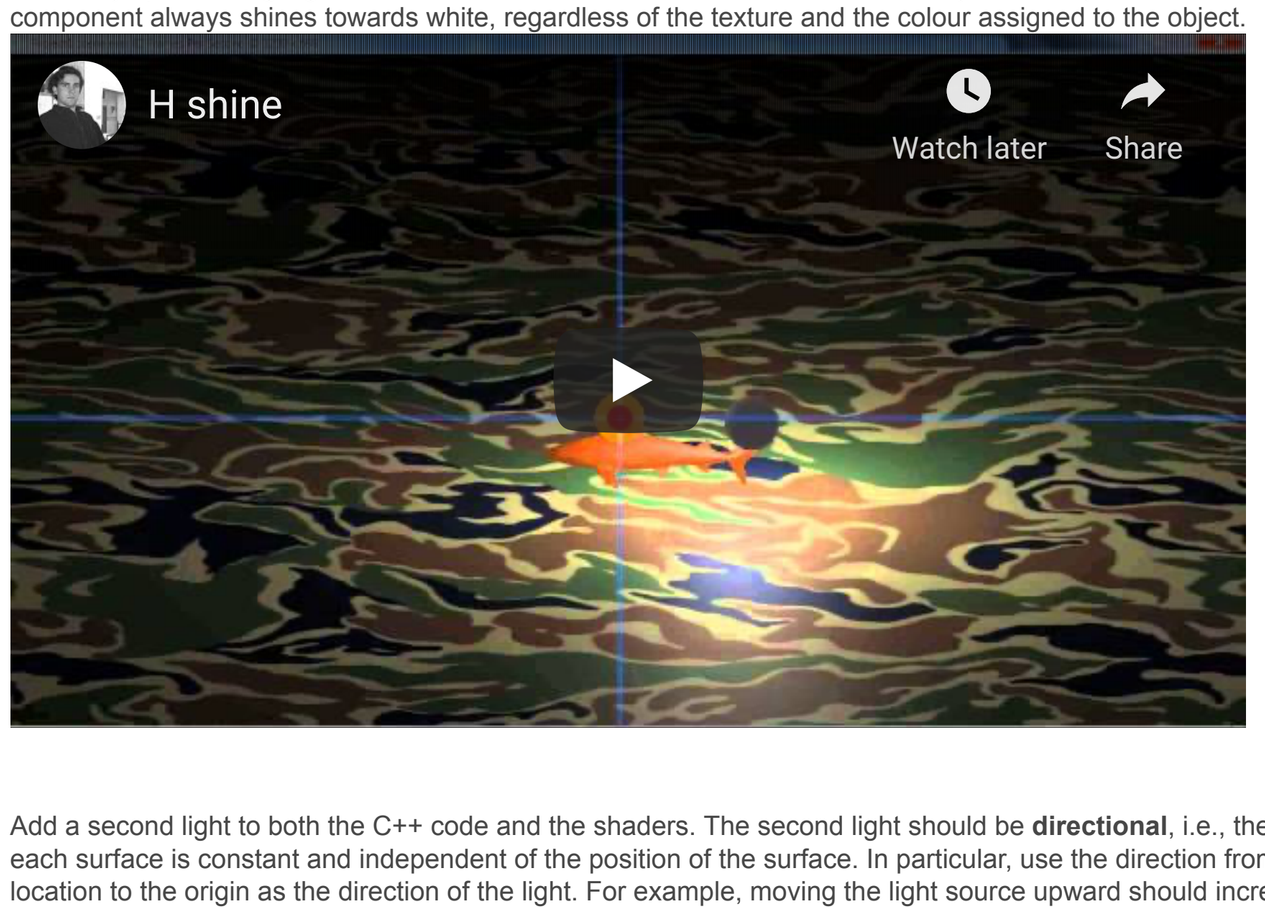
F. Modify the vertex shader so that the light reduces with distance - exactly how it reduces is up to you, but you should aim for the reduction to be noticeable without quickly becoming very dark, similar to the sample solution. This should apply to the first light, but the video shows a slightly different implementation so the second light was chosen from the menu to demonstrate the concept.



G. Comment out the lighting calculations in the vertex shader and perform the lighting calculations in the fragment shader, so that the directions are calculated for individual fragments rather than for the vertices. This should have a very noticeable effect on the way the light interacts with the ground (especially when the ground has a plain texture map), since the ground only has vertices at the corners of a coarse grid.



H. Generally specular highlights tend towards white rather than the colour of the object. Modify the shader code so that the specular component always shines towards white, regardless of the texture and the colour assigned to the object.



I. Add a second light to both the C++ code and the shaders. The second light should be **directional**, i.e., the direction that the light hits each surface is constant and independent of the position of the surface. In particular, use the direction from the second light's location to the origin as the direction of the light. For example, moving the light source upward should increase the y-component of the direction of the light source, making the light source behave like the mid-day sun. Like the first light source, use `sceneObj[2]` to store the coordinates and colours of the second light, and make it similar to the first light, including placing a sphere at the coordinates of the light.

J. In your editor, allow (a) objects in the scene to be deleted, and (b) duplicated. (c) Add a spot light to the scene and allow its illumination direction to be changed interactively.