# Assignment 2

## Contents



# Task:

Write a “3D cube pipe” program. Your program will create a series of cubes joining together to form a pipe.

Each cube has identical size and each side of a cube is colored differently. When a cube is first generated, it can be rotated and translated in 3D space. It is also assigned with an integer identifier. In this program, a newly created cube can only be joined with a cube that was created immediately before, i.e., a cube with index i can only be joined to a cube with index i – 1. The user can also specify by which of the six faces the two cubes should be joined.

Your program is also capable of the following tasks:

1. Rotate a cube: Firstly, the user selects a cube to rotate, calling it the “rotating cube”, then every cube created after the “rotating cube” will rotate with it. The rotation is about an axis that is perpendicular to the face which it joins with the previously created cube. The rotation axis also intersects with the center of this face. For example, if a user specifies the “rotating cube” to be cube 6, then every cube with number higher than 6 will also rotate with it, about an axis that is perpendicular to the face joining cube 5 and 6, and the axis also intersects with the center of that face.
2. Delete a cube: When a user specifies the deletion of a cube i, then the cubes i -1 and i + 1 join together.

# Solution Approach:

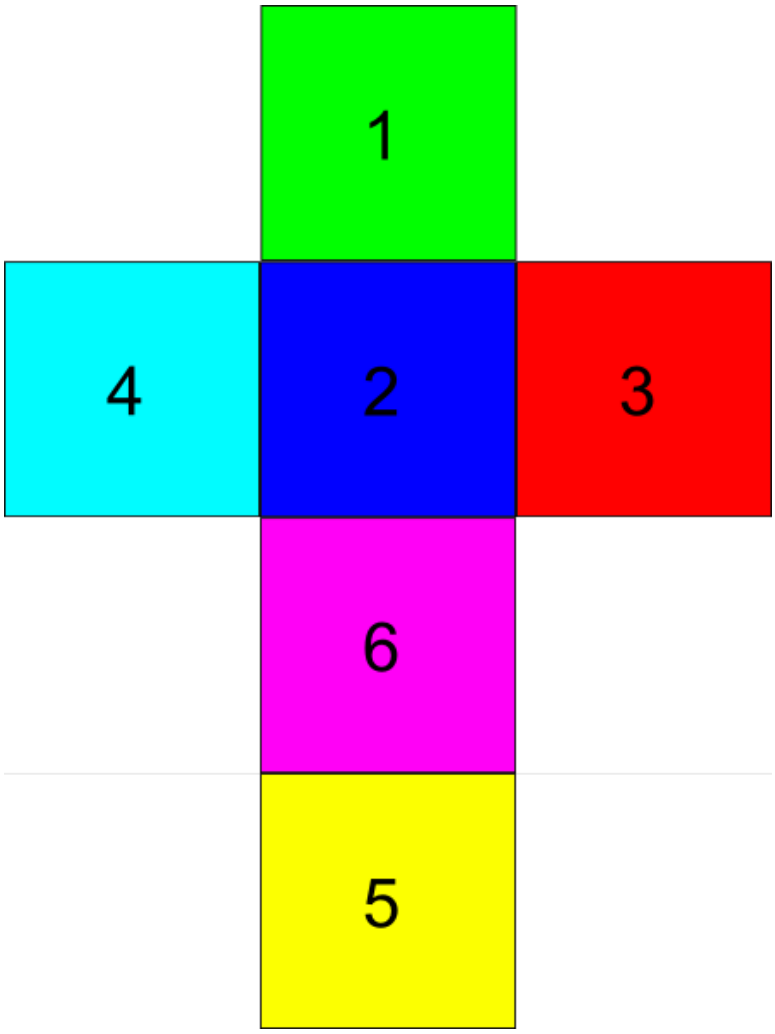
The task was broken down into a set of requirements as follows:

### *Your program will create a series of cubes joining together to form a pipe.*

"Joining together" was interpreted to mean that cubes would be drawn next to each other to give the illusion of being connected. A "pipe" is a connected structure that doesn't branch, so cubes can only be added the the head or the tail of the pipe.

### *Each cube has identical size and each side of a cube is colored differently.*

Each cube would be identical in that while the faces were coloured differently, each instance of a cube would have the same colour on the same relative face. Therefore one cube mesh is defined and multiple instances are drawn. The cubes are coloured according to the following diagram (the number on each face is the key that will create a cube attached to that face of the previous cube).



## *When a cube is first generated, it can be rotated and translated in 3D space.*

The program allows two methods of creating cubes. N will create a detached cube, used as the head of a pipe. This cube can be rotated and translated along any axis, giving it six degrees of freedom. Cubes added to an existing pipe (with the keys 1-6) can only be rotated about the axis normal to the face they are connected to, giving them only one degree of freedom. The head of the pipe can be translated by holding down the left mouse button and moving the mouse/scrolling with the mouse wheel. Holding down the right mouse button will rotate the head of the pipe.

When a cube is initially drawn it is set as the active cube.

## *It is also assigned with an integer identifier.*

From reading the criteria that the integer identifier was only used to specify what cubes are connected to each other, and which cubes would be joined upon a deletion. Therefore a doubly linked list is used to store the boxes and provide ordering information instead of an integer identifier.

To identify which cube the user is able to modify one cube is set as the “active” cube and is drawn as a wireframe. This does make it slightly harder to see which face has which colour. The unselected cubes are drawn as filled polygons.

## *In this program, a newly created cube can only be joined with a cube that was created immediately before, i.e., a cube with index i can only be joined to a cube with index i – 1.*

The program only adds cubes to the last cube created. No branches are allowed, and no cubes can be added to a pipe which is not the last pipe. (While this would be possible, and desirable, it's explicitly forbidden in the requirements and as such is not implemented.)

## *The user can also specify by which of the six faces the two cubes should be joined.*

The numbers 1-6 specify the face of the tail cube that the new cube should be joined to. Each cube has the same basic orientation, while their rotation may be adjusted (cubes are created with the same face facing upward). If the tail cube is not the head of the pipe, then one face will not be available for a new cube to join to, and any attempts at adding a cube to that face will silently fail.

## *Rotate a cube*

The user can select which cube is the active cube (the white cube) by pressing + or – (moving through a pipe), Page Up or Page Down (moving to the previous/next pipe), Home or End (moving to the first/last pipe). If the active cube is the head of the pipe, rotating the mouse wheel without a button held down will do nothing. If the active cube is not the head of the list, rotating the mouse wheel will rotate the cube about it's axis.

Holding down the right mouse button and moving the mouse or rotating the mouse wheel will rotate the head of the pipe (and as such, the entire pipe will rotate with it).

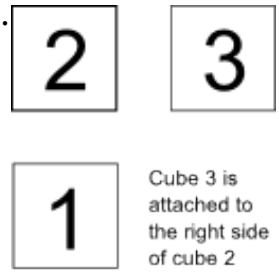
Cubes will rotate around their centre.

Due to the way OpenGL uses matrix stacks to perform transformations, a pipe will be drawn with the following method:

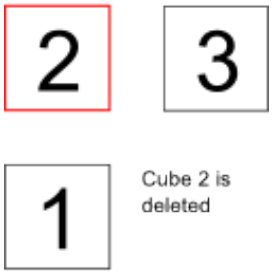
1. Draw the tail cube.
2. Rotate the cube about its axis.
3. Translate the cube along its axis.
4. Draw the previous cube.
5. Rotate the scene about its axis (which will also rotate all other cubes drawn so far).
6. Translate the scene along its axis (which will also translate all other cubes drawn so far).
7. Repeat steps 4 – 6 until we reach the head cube.
8. Draw the head cube.
9. Rotate the scene by the head cubes rotation matrix.
10. Translate the scene by the head cubes position vector.

## *Delete a cube*

The user can delete the active cube with the Backspace or Delete keys. If the active cube is the head of a pipe, all translation and rotation information will be lost and the pipe will move back to the origin. If the active cube is anywhere else in the pipe it will be removed and the next cube will attempt to attach itself to the previous cube on the same face. (See diagram).



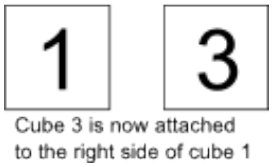

If deleting the active cube would cause three cubes to be joined at the same face, it will silently fail and no action will be performed. In this event the user needs to press Shift + Backspace or Shift + Delete and the tail of the pipe will be dropped, starting from the active cube.




If the user deletes the head of a pipe, and it turns out to be the only element in the pipe, the pipe is deleted and the previously created pipe is made active.

## *Data structures*

The program stores a list of pipes (using the stl list data structure) internally. Each element of this list is a pointer to the head of a pipe, which implements a doubly linked list. A list was chosen as random access is not required, and it allows easy iteration through elements in both directions.



# Limitations and Bugs:

The number of cubes that can be created is limited only by the stack size. (A conservative guess would place the upper limit at approximately 30 000 cubes.)

There are no known bugs in the program.

# Structure of the source code:

Please see the doxygen generated documentation in the Assignment 2/Documentation/html directory.

## User guide:

To add boxes:

| Key | Description |
|---|---|
| N | Creates a new box, not attached to any previous box. |
| 1 | Adds a box to the blue face of the previous box. |
| 2 | Adds a box to the magenta face of the previous box. |
| 3 | Adds a box to the green face of the previous box. |
| 4 | Adds a box to the yellow face of the previous box. |
| 5 | Adds a box to the red face of the previous box. |
| 6 | Adds a box to the cyan face of the previous box. |

To change which box is active box (the white box):

| Key | Description |
|---|---|
| + or = | Move toward the tail (the last box) of the current pipe. |
| - | Move toward the head (the first box) of the current pipe. |
| Page Up | Select the previous pipe. |
| Page Down | Select the next pipe. |
| Home | Select the first pipe. |
| End | Select the last pipe. |

To transform the active box:

| Key | Description |
|---|---|
| Scroll wheel up | Rotate the box clockwise. |
| Scroll wheel down | Rotate the box counter-clockwise. |
| Right Mouse Button | Rotate the head of the pipe while held down (move the mouse and use the scroll wheel to rotate around each axis). |
| Left Mouse Button | Translates the head of the pipe while held down (move the mouse and use the scroll wheel to translate along each axis). |

To delete the active box:

| Key | Description |
|---|---|
| Backspace or Delete | Will remove the box as long as it's removal won't cause a conflict. |
| Shift+Backspace or Shift+Delete | Will remove the box, but if there is a conflict the rest of the pipe (beyond that box) will be deleted with it. |

To move the camera:

| Key | Description |
|---|---|
| W or Up | Move the camera forward. |
| S or Down | Move the camera backward. |
| A or Left | Move the camera left. |
| D or Right | Move the camera right. |
| Space | Move the camera up. |
| Ctrl | Move the camera down. |
| O | Stores the current position as a control point. |
| Tab | Toggles between orbit (observer) mode, where the camera moves along a path defined by the control points while looking at the origin, and free-form movement. |

To rotate the camera:

| Key | Description |
|---|---|
| Middle Mouse Button | Rotates the camera while the button is held down (move the mouse to pitch and yaw). |

# Source Code:

| Filename | Purpose |
|---|---|
| main.cpp | Program entry point, input cycle, update cycle, render cycle. |
| Box.h<br>Box.cpp | Class defining the cube elements used to construct the pipe. |
| Camera.h<br>Camera.cpp | Class defining a user controlled camera. |
| DynamicCamera.h | Abstract class for an up-datable camera. |
| ThirdPersonCamera.h<br>ThirdPersonCamera.cpp | Class that turns to face the target point every update cycle. |
| ElasticCamera.h<br>ElasticCamera.cpp | Class that moves along a B-Spline defined by a list of control points. |
| ElasticThirdPersonCamera.h<br>ElasticThirdPersonCamera.cpp | Class combining the functionality of the two classes above. |
| Vector3.h<br>Vector3.cpp | Class encapsulating common vector operations and a (mathematical) vector data structure. Used to define the translation and rotation of the cubes, as well as store the position and orientation of the camera. |
| Quaternion.h<br>Quaternion.cpp | Class encapsulating common Quaternion operations and a Quaternion data structure. Used to rotate the camera and MasterBoxes. |

# Additional Libraries:

| Library | Purpose |
|---|---|
| SDL | Window manager (provides roughly the same functionality as glut, more powerful and portable). |
| Boost | Smart pointers (to manage memory allocation/deallocation), Time (provides a portable timer for use during the update cycle). |

# Other Files:

| Filename | Purpose |
|---|---|
| Assignment 2.odt | OpenOffice.org text version of the assignment documentation. |
| Assignment 2.doc | Microsoft Word version of the assignment documentation. |
| Assignment 2.pdf | Portable Document Format version of the assignment documentation. |
| cubemap.png | Unfolded cube net showing the face colours and corresponding key press. |
| deletion.png | Image showing the process when deleting a cube. |
| output.png | Example output of the program. |
| cubemap.svg | Scalable Vector Graphics version of cubemap.png (Outdated) |
| deletion.svg | Scalable Vector Graphics version of deletion.png |
| html/index.html | Doxygen generated documentation of all source code files in html format. |
| latex/ | Doxygen generated documentation of all source code files in LaTeX and PDF format. |

## Example Output: