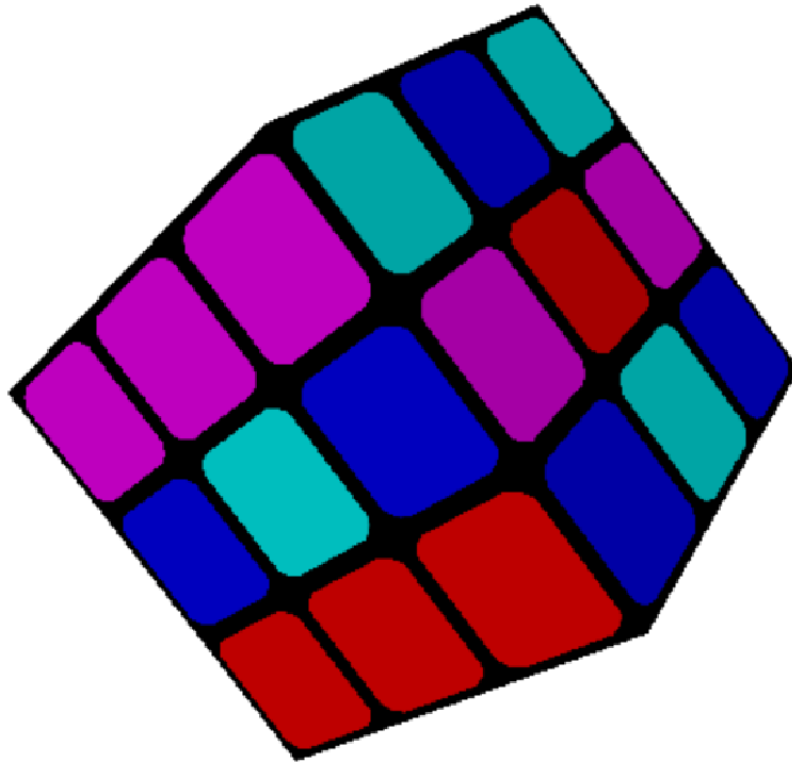# Assignment 3 – Rubik's Cube

Overview:

In this assignment you will render a dynamic Rubik's cube using **OpenGL**..

Part 1 – building the Rubik's cube scene:

1. Create a **Perspective View Camera** with a **FOV Angle** of 45.
2. Build a **Cube** mesh with the **plane.png Texture** to render a single cube.
   See what happens when you translate and rotate the cube.
3. For building the Rubik's cube you will need to duplicate the cube 26 or 27 times and translate each cube to its initial position.

Part 2 – data structures:

1. Build data structures that will hold information about the current position of each cube on the full cube based on their shape index.
   You will need to know which of the cube you need to rotate each time you rotate a wall of the Rubik's cube.
2. You may build other data structures to hold for each cube its current rotations and translation.
   This data structure will save the transformations you did for each cube.
   (But it is possible to solve this task without them).
3. Build data structure (class) to hold all the Rubik's cube data.
   (It is also possible to store all the data on the available classes in the engine if you find it easier).

Part 3 – rotations in a row (the difficult part):

1. Keyboard keys callback function:
    a. "R" – press state for **right** wall rotation (90 degrees clockwise).
    b. "L" – press state for **left** wall rotation (90 degrees clockwise).
    c. "U" – press state for **up** wall rotation (90 degrees clockwise).
    d. "D" – press state for **down** wall rotation (90 degrees clockwise).
    e. "B" – press state for **back** wall rotation (90 degrees clockwise).
    f. "F" – press state for **front** wall rotation (90 degrees clockwise).
    g. " " (Space) – press state for **flipping** rotation direction.
        (from clockwise to counterclockwise or vice versa).
    h. "Z" – press state: dividing rotation angle by 2 (until minimum of 45).
    i. "A" – press state: multiply rotation angle by 2 (until maximum of 180).
2. Mouse keys callback function:
    a. Moving while holding **left button** will rotate the **cube**:
        Left to right movement will rotate the cube around the **global Y** axis and up to down movement will
        rotate it around **global X** axis.
    b. Moving while holding the **right button** will move the **camera** up, down, left or right.
    c. **Scrolling** up and down will move the camera along the **Z axis** backward and forward
        (Make sure that the cube is moving backward and forward from the current view).
3. Arrow keys callback function:
    a. If you <mark>DON'T</mark> plan to implement the **Rubik's cube in dynamic sizes bonus**,
        set the arrow key to rotate the Rubik's cube around the **X axis** and **Y axis** of the scene.
    b. However, if you <mark>DO</mark> plan to implement this **bonus**,
        use the arrow keys and the "I", "O" keys to change the **center of rotation** for the cube to enable
        rotation of all the Rubik's cube walls
        (4 arrow keys and 2 letter keys for 6 possible movements on the **X axis**, **Y axis** and **Z axis**).
4. Color Picking:
    a. The Picking Mode should get enabled and disabled by pressing the "P" key.
    b. Picking a specific cube with the mouse, to **translate** and **rotate** it, should be possible.
    c. When translating the picked cube with the mouse **right** click, the cube should stay under the mouse.
        (Implement the **Z-Buffer** formula to make it work).
    d. When rotating the picked cube with the mouse **left** click, the cube should be rotated according to the
        current camera view.

Part 4 – cube functionality:

1. The cube structure should never "break" after multiple different wall rotations.
2. Some rotations should be locked when there are walls rotated in 45 degrees on the cube
    (like in a real Rubik's cube)

Part 5 – bonuses:

1. **3 bonus points** for implementing animation for each wall rotation.
2. **5 bonus points** for implementing a working Rubik's cube in dynamic sizes (2x2x2, 4x4x4, 5x5x5, and bigger...).
3. **10 bonus points** for implementation of random mixer (**3 point**) and efficient solver (**7 points**).
    You must be able to display the mixer and the solver action on the screen and write the mixer and solver
    steps sequence to files mixer.txt and solver.txt.
    You may check the executable file to compare to your results.
    The available keys you can use are: "M" - Mixer and "S" - Solver.

Color Picking Guidance:

You may choose in which way you want to implement the Color Picking method, but here are some suggested steps to do it: by modifying the MouseButtonCallback:

1. Modify the **Shaders** to support Picking Mode, so that in this mode the shape will get their colors only from the uniform **u_Color** parameter.
2. Modify the **MouseButtonCallback** to do as follows:
   a. Clear the window.
   b. Get the: VBO, VAO, Shader and the MVP matrices of all the cubes you created.
   c. Bind the: VBO, VAO and the Shader.
   d. Draw each one of the cubes with a unique color in **u_Color** that will be passed to the Shader.
   e. Use the **glReadPixels** function to read the **color** in the selected pixel and convert the color to the relevant cube index and keep it.
   f. Use the **glReadPixels** function to read the depth in the selected pixel (to use for the Z-Buffer) and keep it.
3. Modify the **CursorPosCallback** to check if a cube was selected, and supported the Translate and Rotate operation based on the selected Mouse Button.

General tips:

1. When calculating the Model Matrix per cube, set the Translate, Scale and Rotate Matrices as you see fit:
   a. rot * trans * scl – For rotations around the global center. (Better for this task)
   b. trans * rot * scl – For rotations around the object center.
2. For the 45 degrees implementation you may find it useful to define 6 locks to control valid and invalid wall rotations.
3. First, try to implement one wall rotation in the Rubik's cube.
4. Plan where each cube supposes to be after every rotation and how you can follow each cube (hint: use index array).
5. Then try to implement a support for multiple different rotations in a row.
   Remember that after each wall rotation the rotated cubes' coordinate system is getting changed.
   Use the equations for **Changing Coordinate Systems** to handle that:

$$Z$$

$$W = (W_x, W_y, W_z)$$

$$u = (u_x, u_y, u_z)$$

$$y$$

$$X$$

$$v = (v_x, v_y, v_z)$$
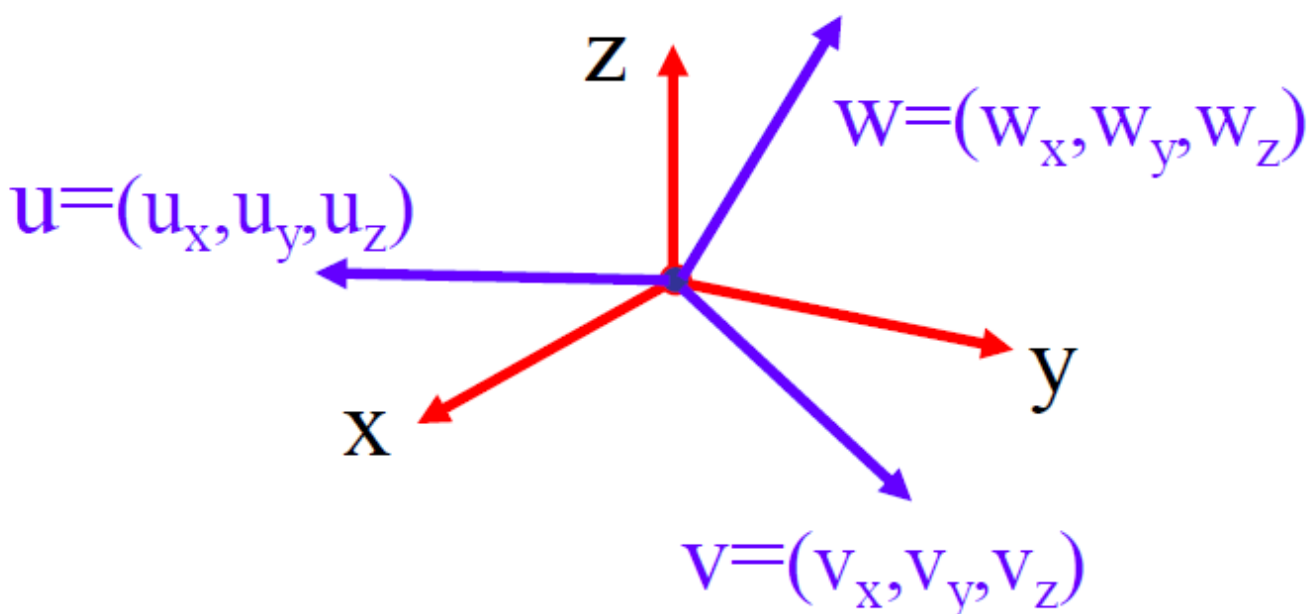
Assignment Score:

- **Sanity** - Rendering the Rubik's cube with a Perspective Camera: **20 points**
- Full support for multiple wall rotations (Clockwise/Counterclockwise) of the Rubik's cube: **20 points**
- Option to 180 degrees rotations: **5 points**
- Option to 45 degrees rotations (including locking the wall rotations when needed): **5 points**
- (**Full Rubik's Cube**) Rotation according to camera view: **5 points**
- (**Full Rubik's Cube**) Translation up/down/left/right according to camera view: **5 points**
- (**Full Rubik's Cube**) Scroll in and out according to camera view: **5 point**
- (**Color Picking**) Z-Buffer translations for a selected cube: **5 points**
- (**Color Picking**) Rotations for a selected cube: **5 points**
- Performance: **20 points**
- Bonus: Up to **18 points**

Submission:

Submit zip file with the following files:

1. Link to your **GitHub** repository.
2. **(Optional)** Text, Doc or PDF file with short explanation about the changes you did in the engine (files you change and functions you modify).
   This file will help you and us to understand what changes you have made in the engine to complete the assignment 😊

The zip file name must include your ID numbers as follows: **ID1_ID2.zip**.