3D Transformations

Stefan Lienhard

Practical Overview

Week	Topic	Practical	Homework
1	C++	C++ Tutorial Framework Setup	Framework Setup
2	2D OpenGL	Intro to GLSL Interfacing GLSL / C++ Working with Textures	Triangle Spirals Checkerboard 2D Planet System
3	3D OpenGL	3D Transformations Orthographic Projections Index Buffers	Perspective Projections Virtual Trackball Animated Triangle Grid

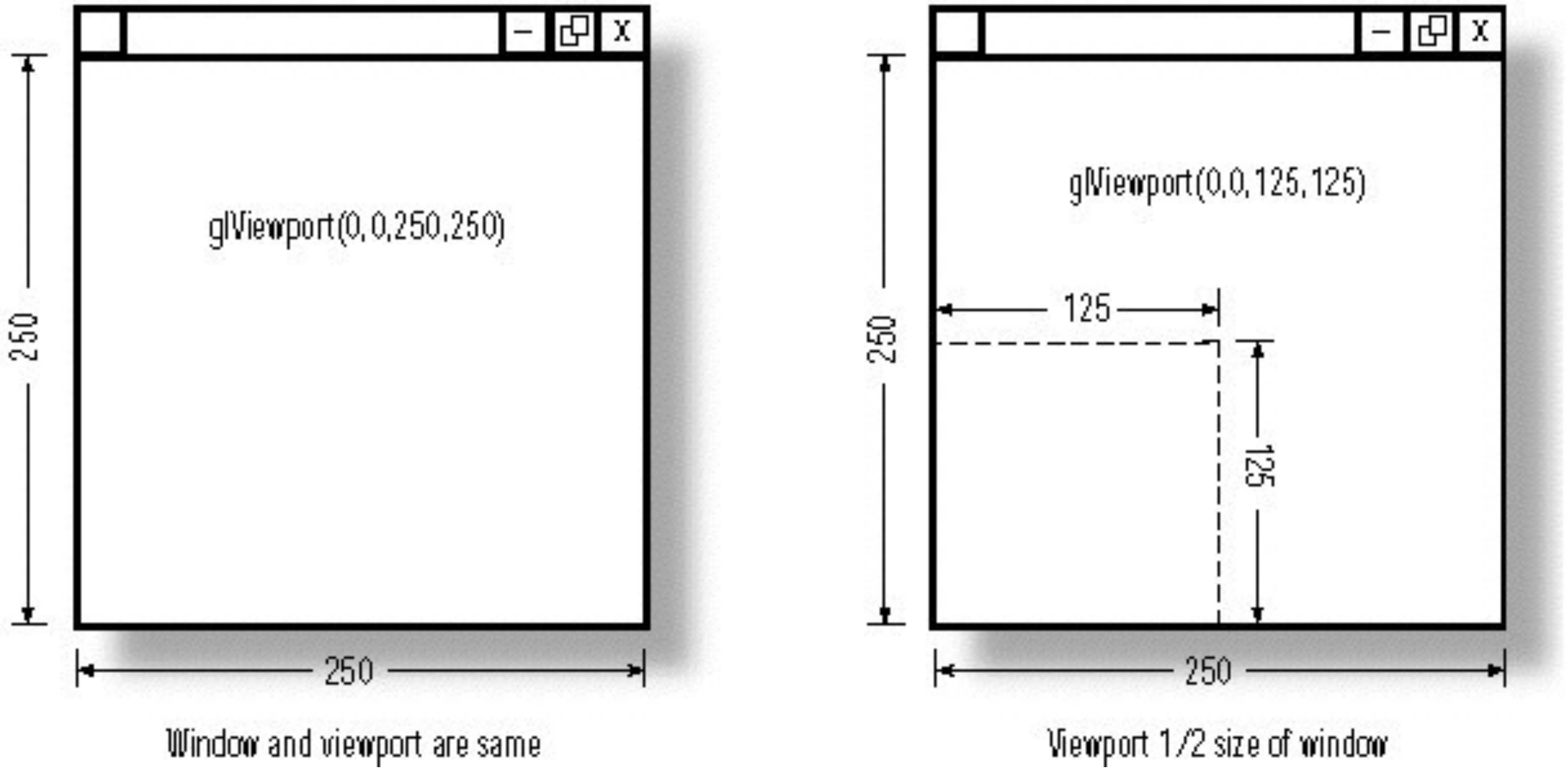
Preliminaries

- Code is in git repository
  - `git clone https://git.epfl.ch/repo/icg15.git`
  - `git pull` (to update already checkout code)
  - if `pull` doesn't work: `git reset --hard HEAD`
    ATTENTION: Backup before doing this
- These slides are in `PDF/Practical-Homework#3.pdf`
- All required changes in the code are marked with `TODO i`, `i` ∈ `(1, …, n)`
- Code for Practical in `lab3_opengl3d`

Window Resizing

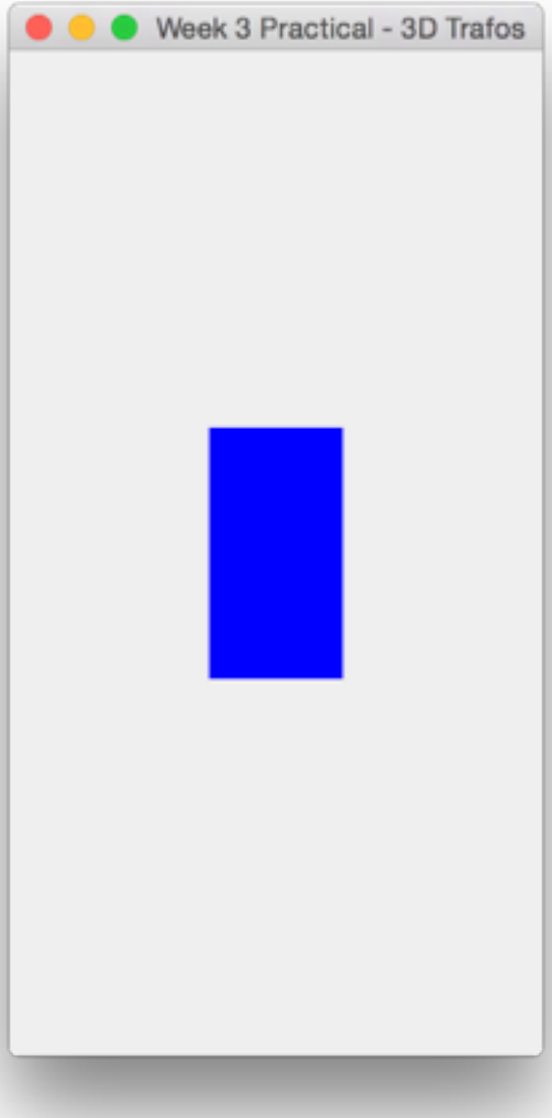
- Call `glViewport(…)` whenever the window is resized to fill the entire window

- `TODO 1` in `main.cpp`

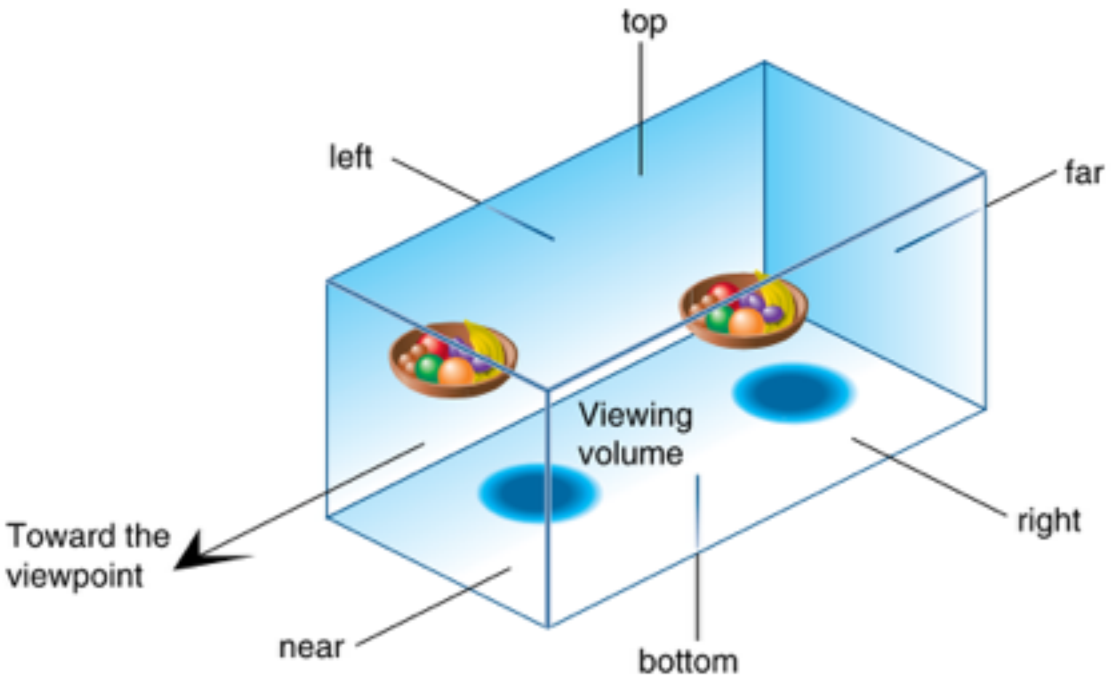


Window and viewport are same

Viewport 1/2 size of window

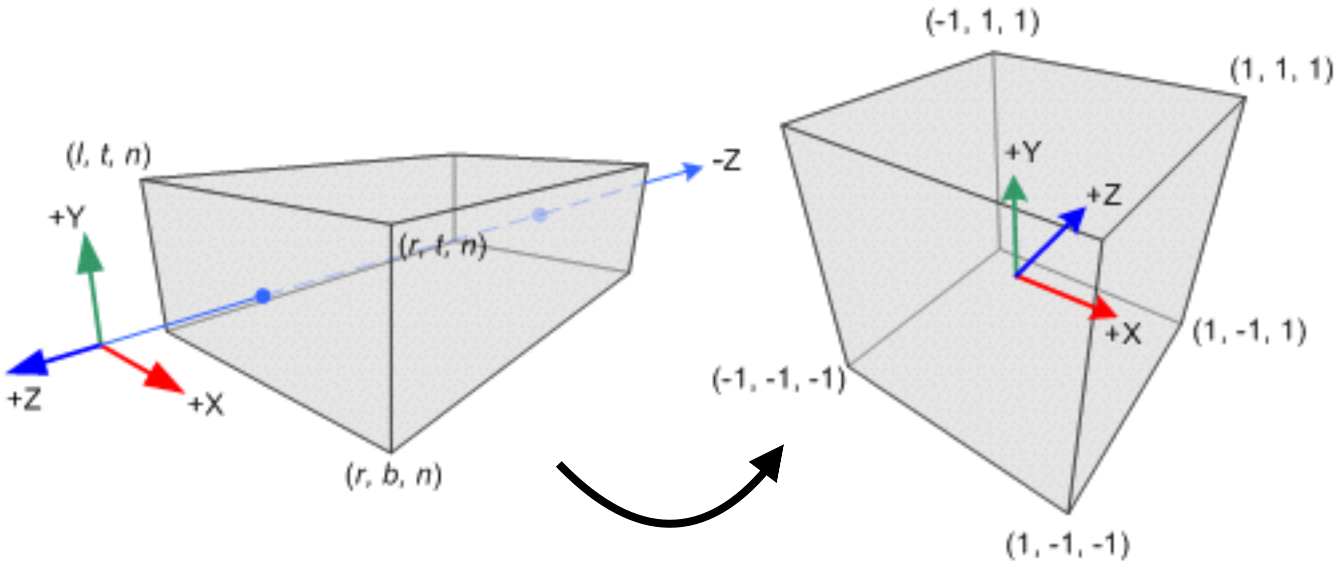


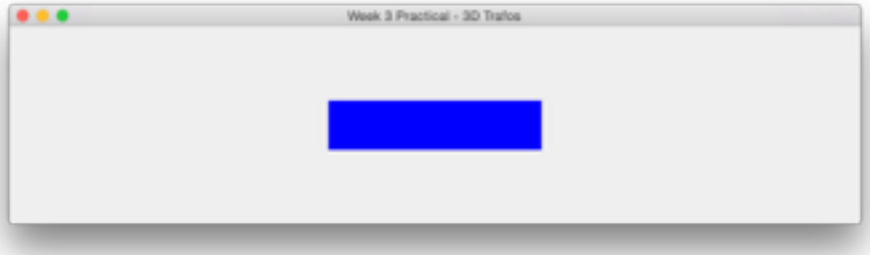
Orthographic Projection



$$\begin{bmatrix} \frac{2}{right-left} & 0 & 0 & -\frac{left+right}{right-left} \\ 0 & \frac{2}{top-bottom} & 0 & -\frac{top+bottom}{top-bottom} \\ 0 & 0 & -\frac{2}{far-near} & -\frac{far+near}{far-near} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- http://www.songho.ca/opengl/gl_projectionmatrix.html#ortho





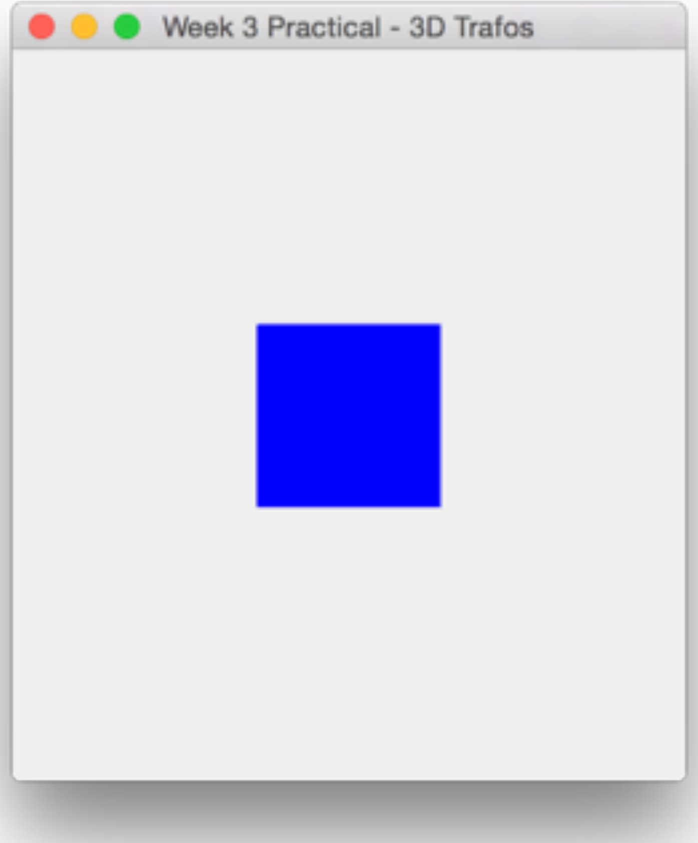
Without a properly set up projection the blue square gets distorted

Orthographic Projection

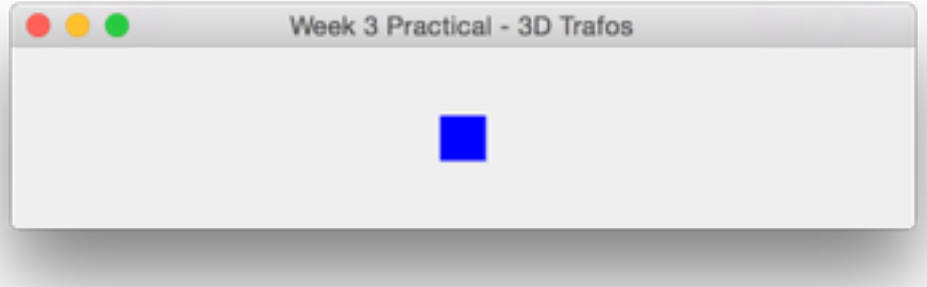
- Setup an orthographic projection matrix
- `TODO 2` in `main.cpp`
- Make sure that

$$\frac{right - left}{top - bottom} = \frac{window\quad width}{window\quad height}$$

  - Fix *top* = 1.0, *bottom* = -*top*
  - Calculate remaining variables with given equation

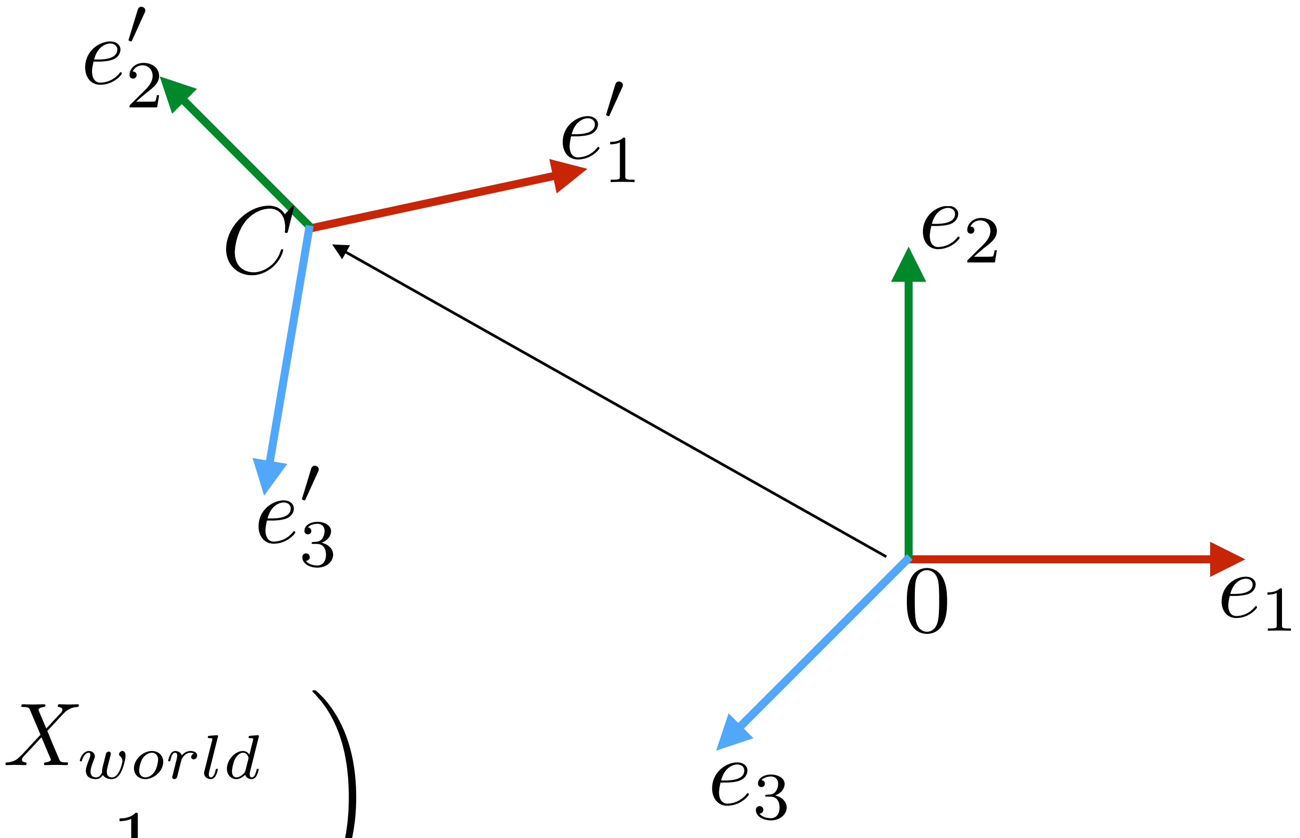


The square stays square when resizing the window



LookAt Transformation

- Transform point *X* from world coordinates into camera coordinates



$$X_{world} = RX_{cam} + C$$

$$\downarrow$$

$$X_{cam} = R^T X_{world} - R^T C$$

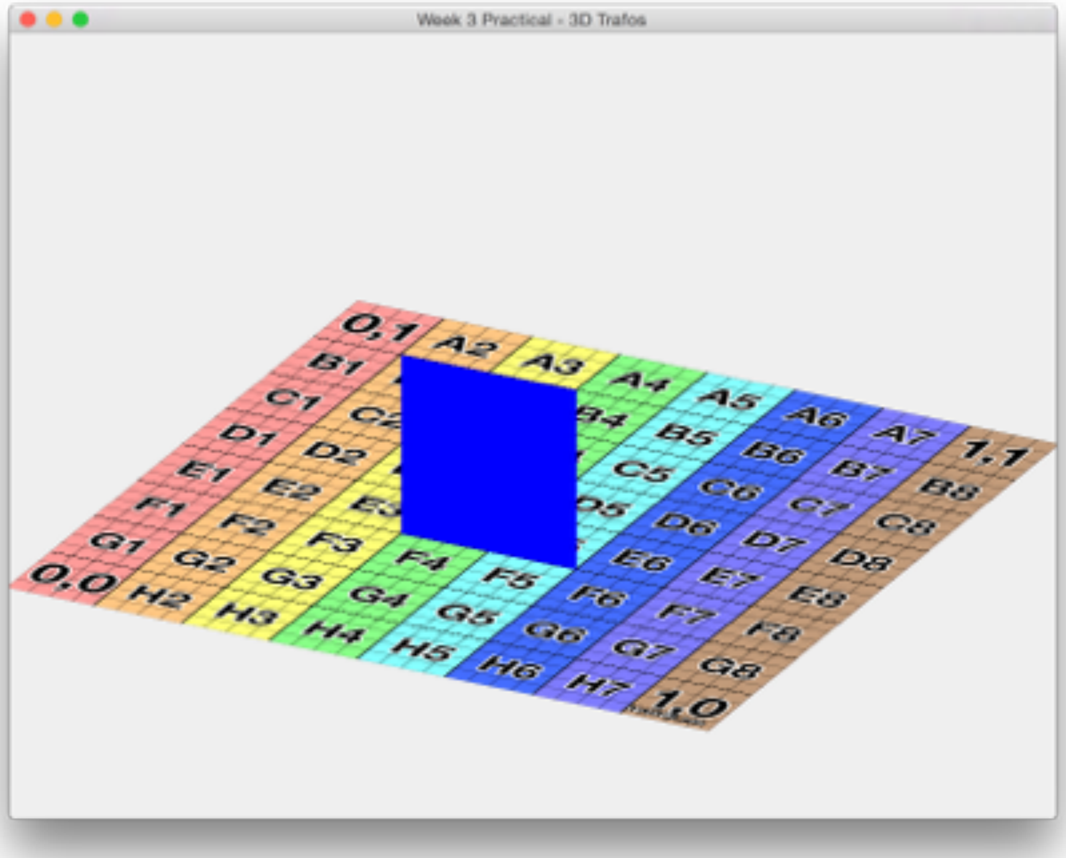
$$\begin{pmatrix} X_{cam} \\ 1 \end{pmatrix} = \begin{bmatrix} R^T & -R^T C \\ 0 & 1 \end{bmatrix} \begin{pmatrix} X_{world} \\ 1 \end{pmatrix}$$

$$R = \begin{bmatrix} e'_{1x} & e'_{2x} & e'_{3x} \\ e'_{1y} & e'_{2y} & e'_{3z} \\ e'_{1z} & e'_{2z} & e'_{3z} \end{bmatrix}$$

$R^{-1} = R^T$, since e'_1, e'_2, e'_3 are orthonormal

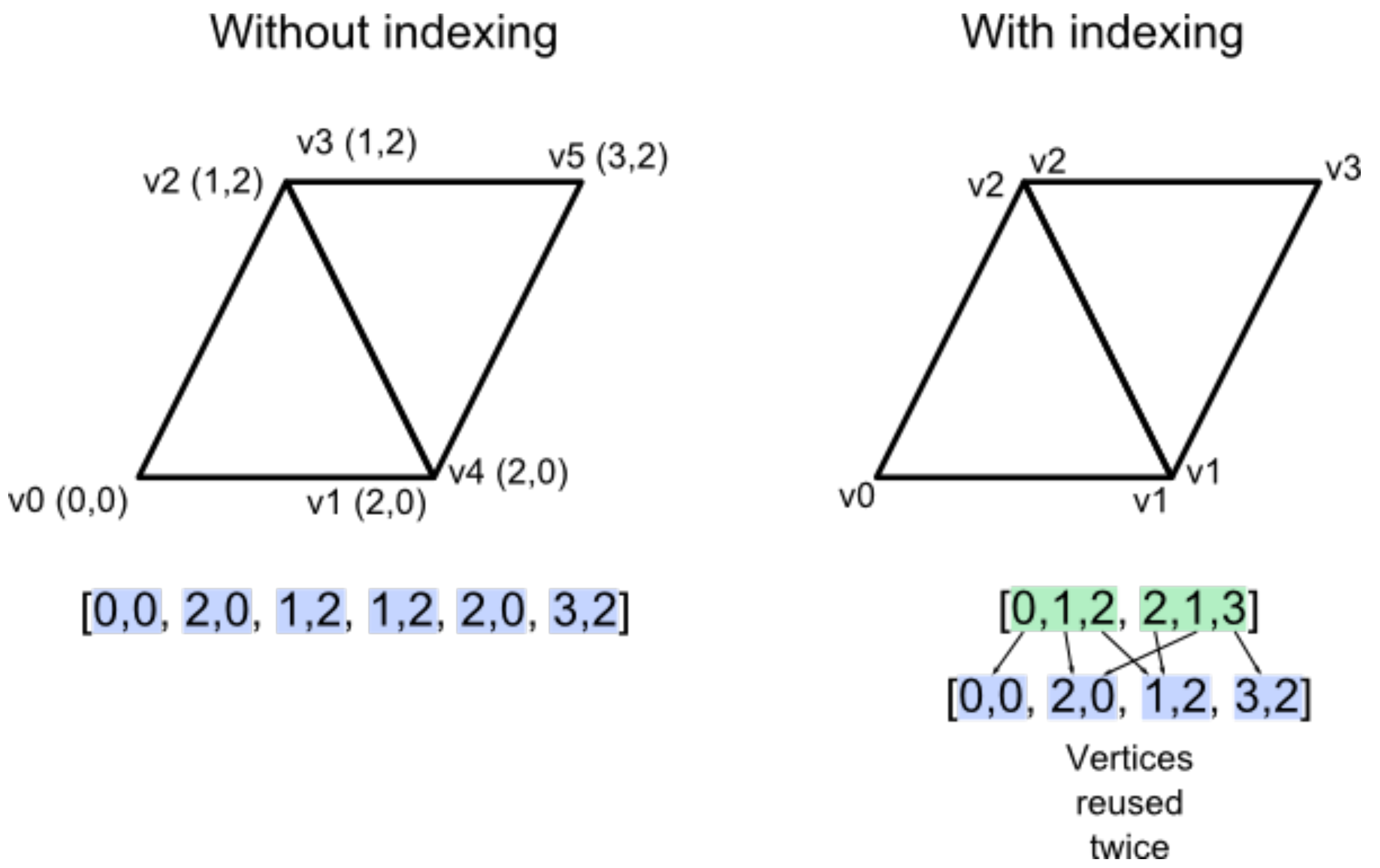
LookAt Transformation

- Implement
  `LookAt(vec3 eye, vec3 center, vec3 up)`
  - `eye` camera position
  - `center` position that you look at
  - `up` camera's up vector
- `TODO 3` twice in `main.cpp`



Cube with Index Buffers

- Each triangle defined by 3 indices into vertex buffer
  - Allows to use vertex positions several times



Without indexing

[0,0, 2,0, 1,2, 1,2, 2,0, 3,2]

With indexing

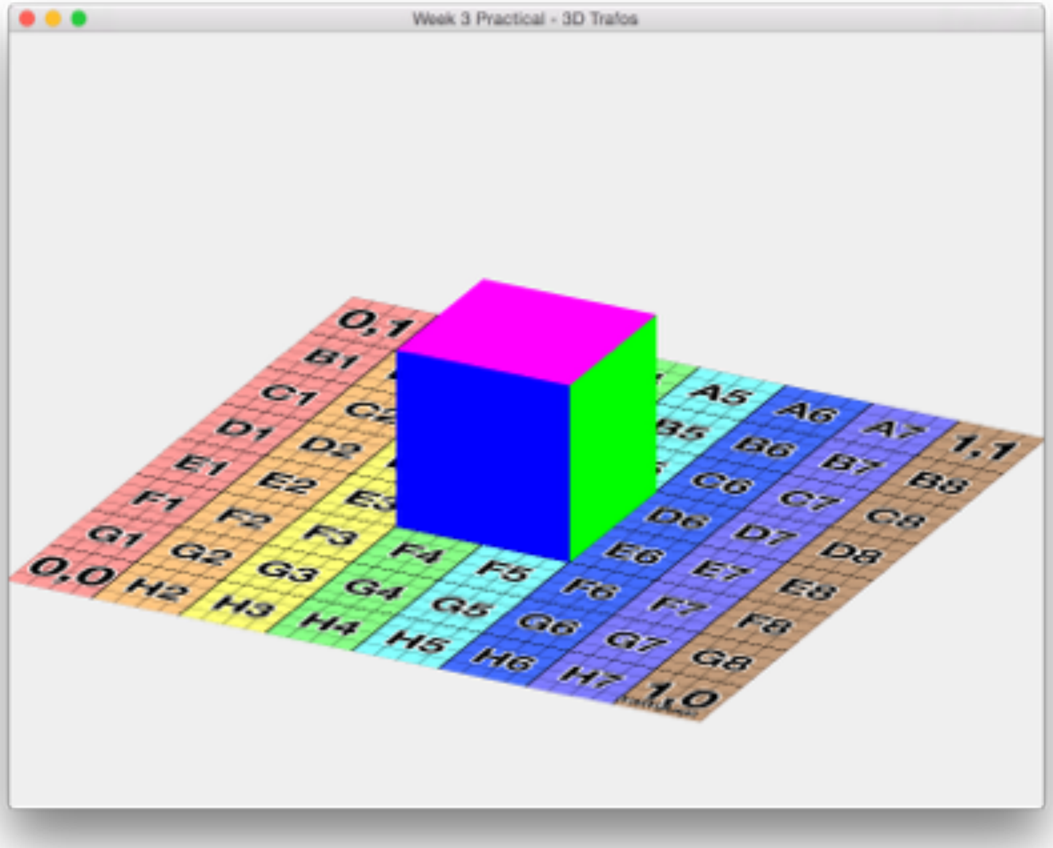
[0,1,2, 2,1,3]

[0,0, 2,0, 1,2, 3,2]

Vertices reused twice

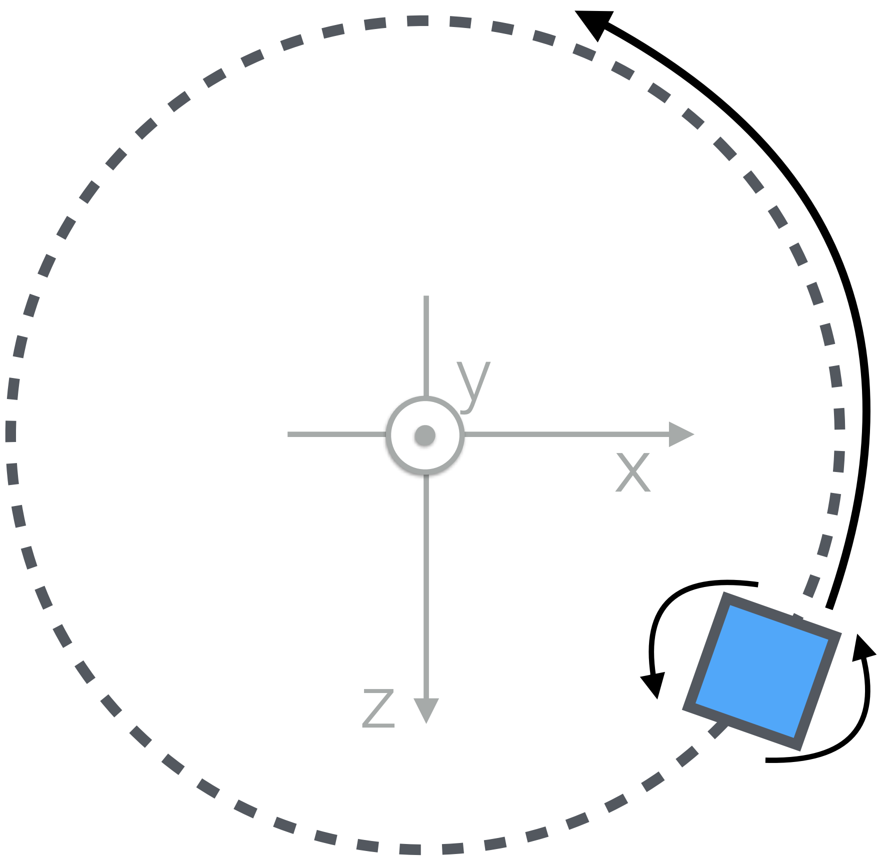
Cube with Index Buffers

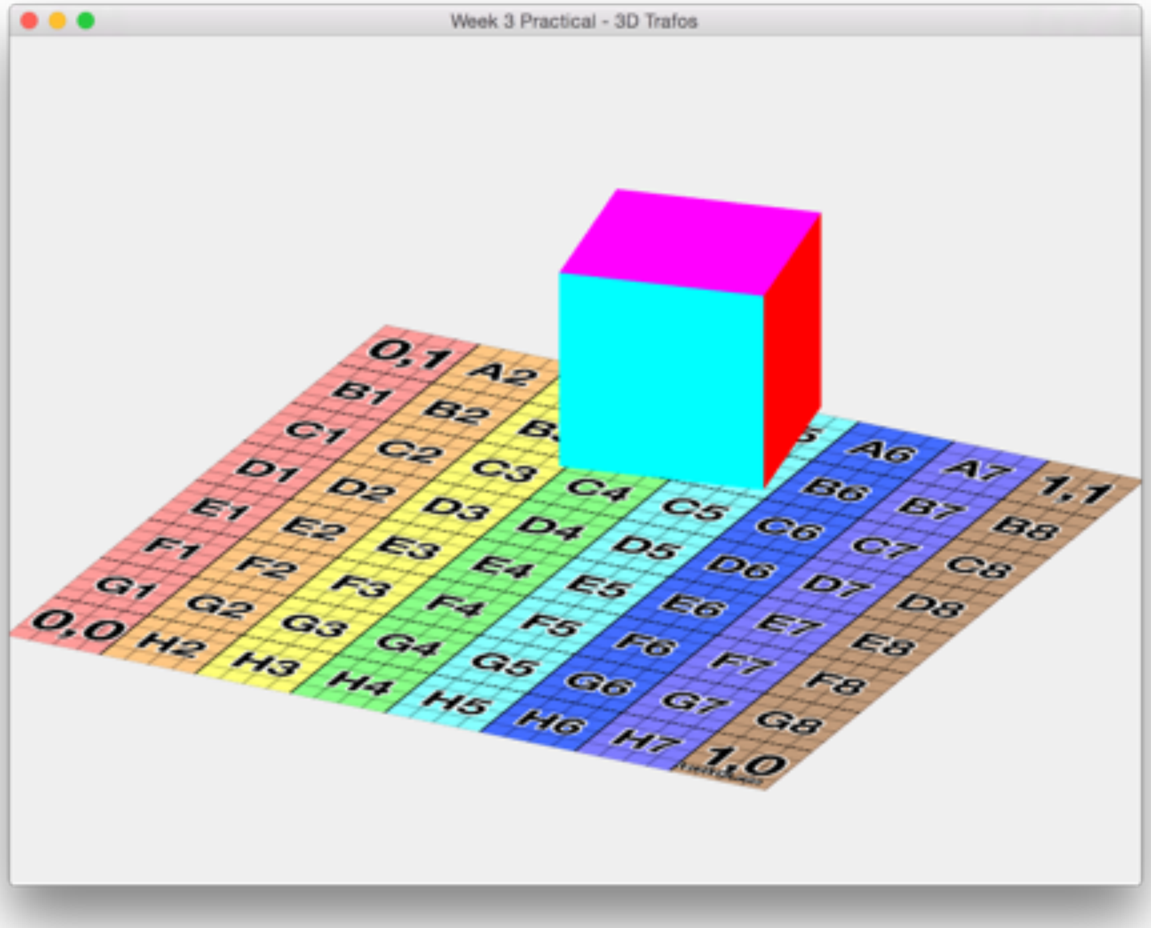
- Index Buffers
  - Need `GL_ELEMENT_ARRAY_BUFFER` (see code)
  - Render with `glDrawElements(…)` instead of `glDrawArrays(…)` (see code)
- Given 8 vertices of a cube and indices for first face, complete the 5 remaining faces `TODO 4` in `cube.h`



Animation of the Cube

- Cube rotates around its own axis while the cube also rotates around the origin





- `TODO 5` in `main.cpp`

Homework 3

Homework 3

Homework 3

- 6 points in total
    - 2 points perspective projection
    - 2 points trackball
    - 2 points animated triangle grid
- Same submission criteria as last week
  (include all source code and binaries that run in INF3, no build folders, optionally readme.txt; -0.5points for including build files, -1.5points for non-running binaries)
- Moodle submission deadline March 12 at 10am
- Homework code in **`hw3_trackball`**

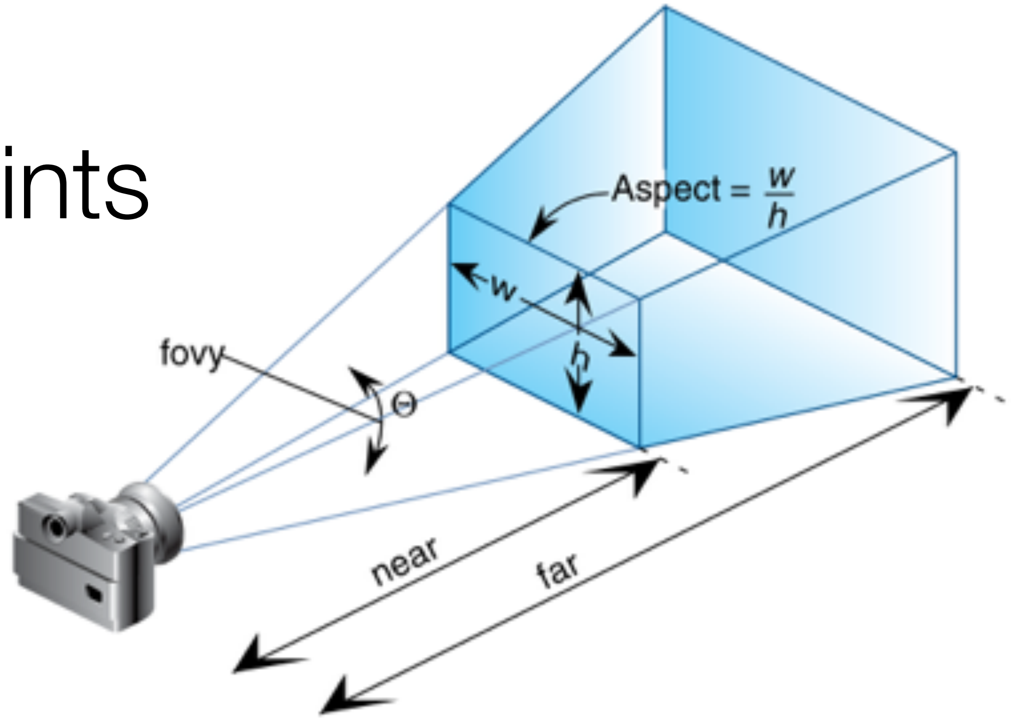
1. Perspective Projection

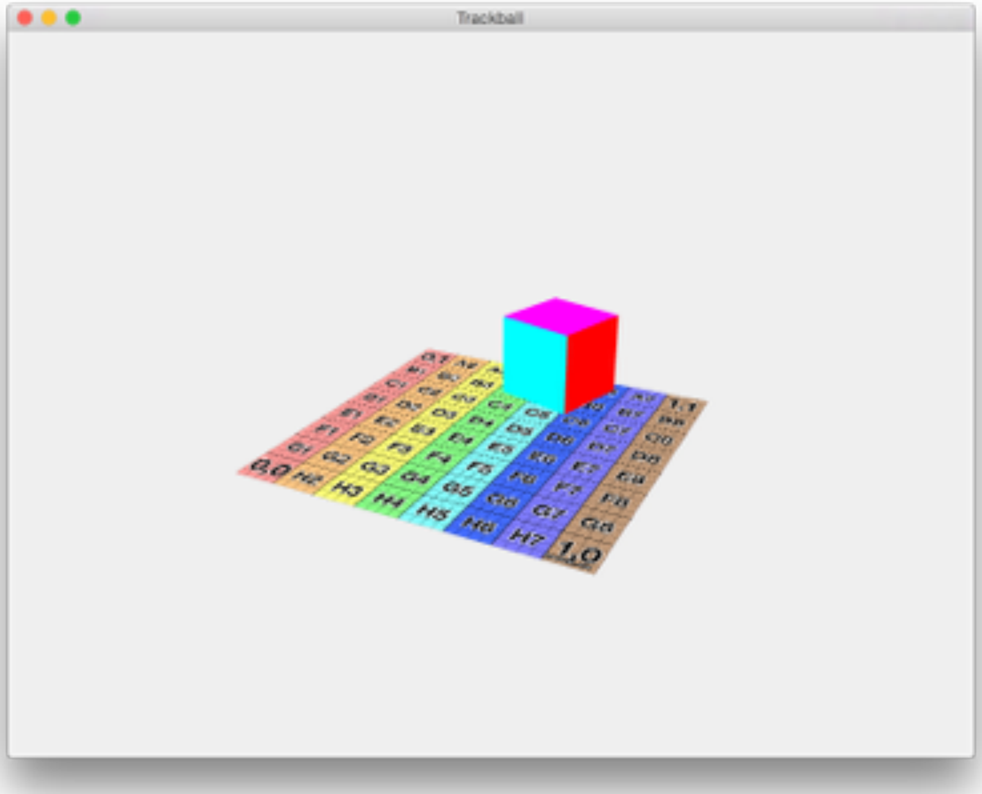
- Build a perspective projection matrix as function of field of view, aspect ratio, and near and far distances

- http://www.songho.ca/opengl/gl_projectionmatrix.html#perspective

- `TODO 1` twice in `main.cpp`

- 2 points



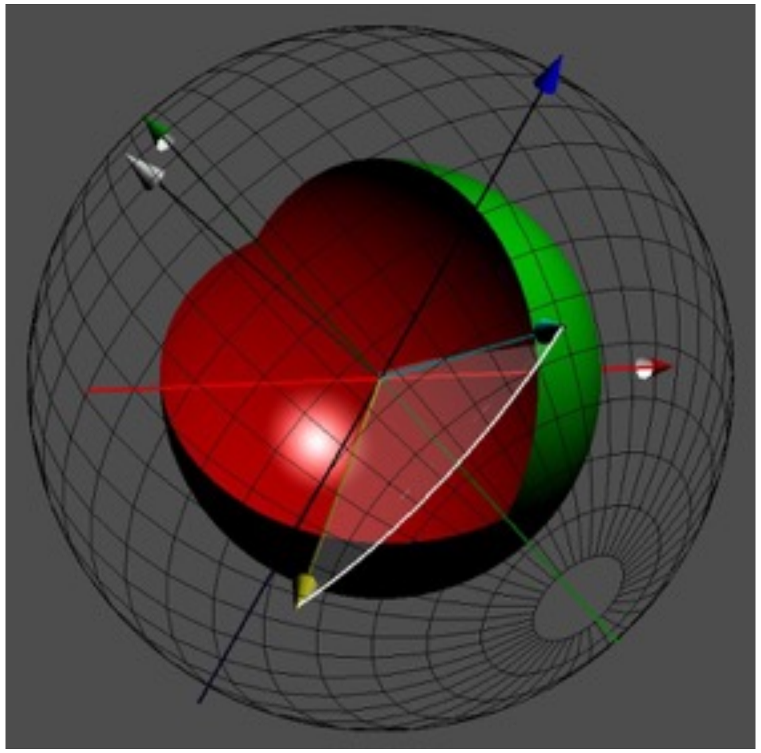


2. Trackball

- Intuitive user interface for complex 3D object rotation via a simple, virtual sphere

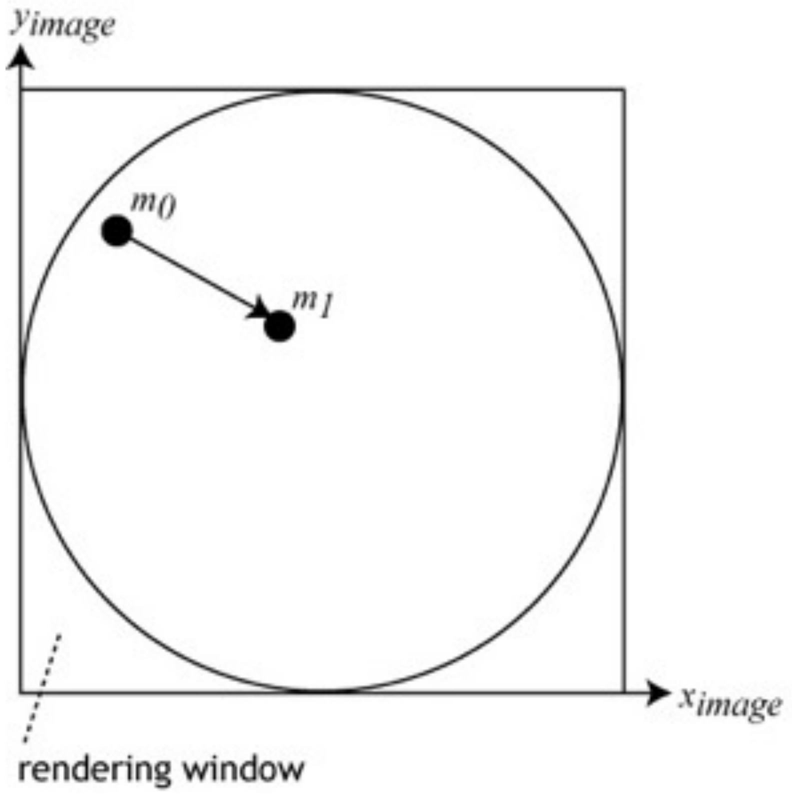
- Put virtual sphere around the object and allow user to rotate it by clicking the left mouse button and dragging the cursor around

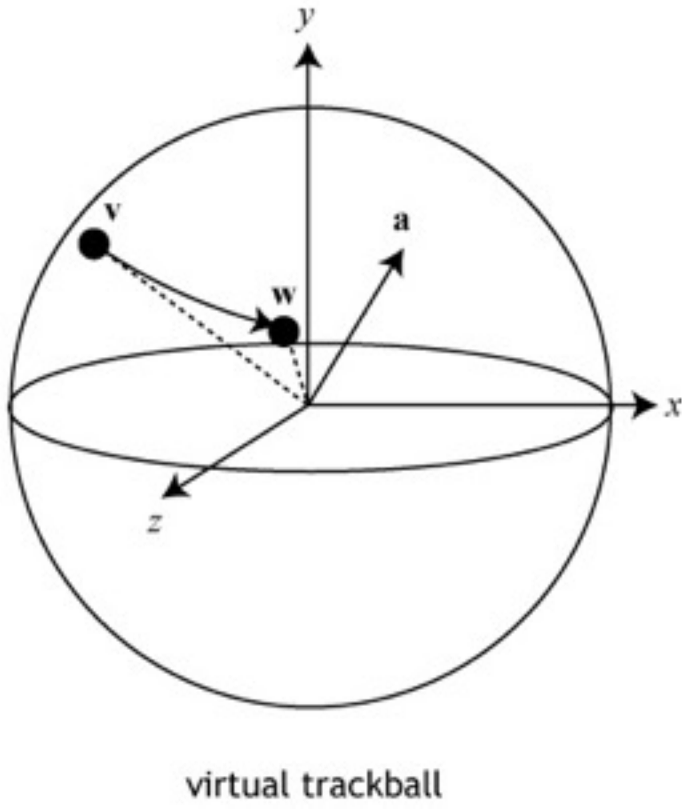
- For an example see MeshLab http://meshlab.sourceforge.net/



2. Trackball

- Map screen space position onto the virtual sphere



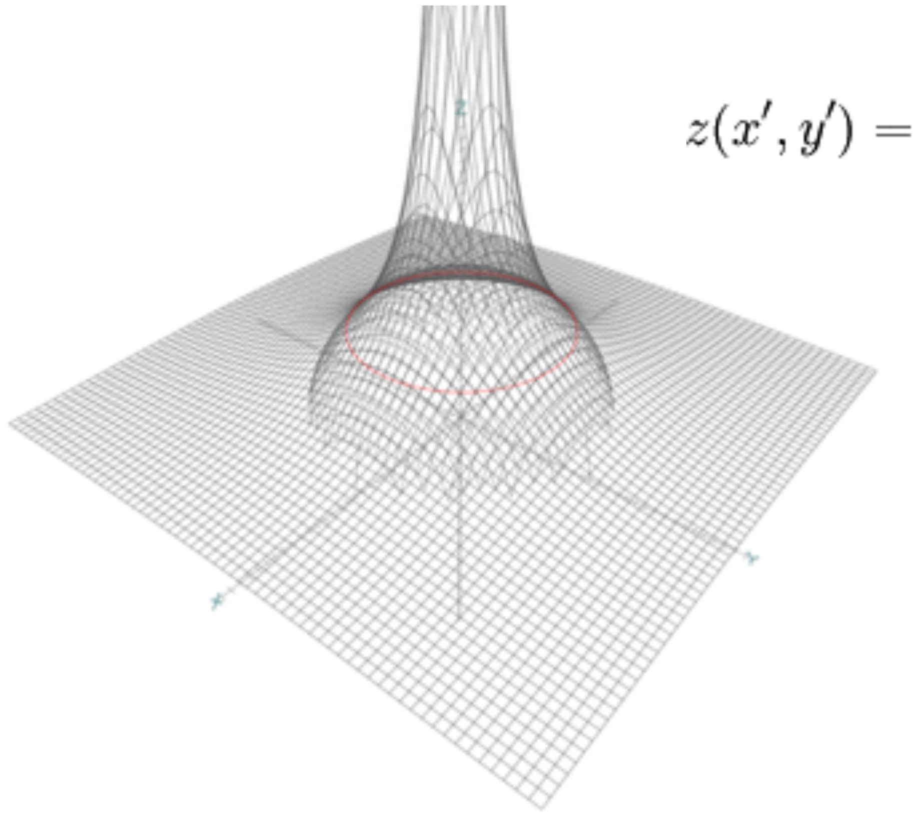


$$z(x', y') = \sqrt{r^2 - (x'^2 + y'^2)}$$

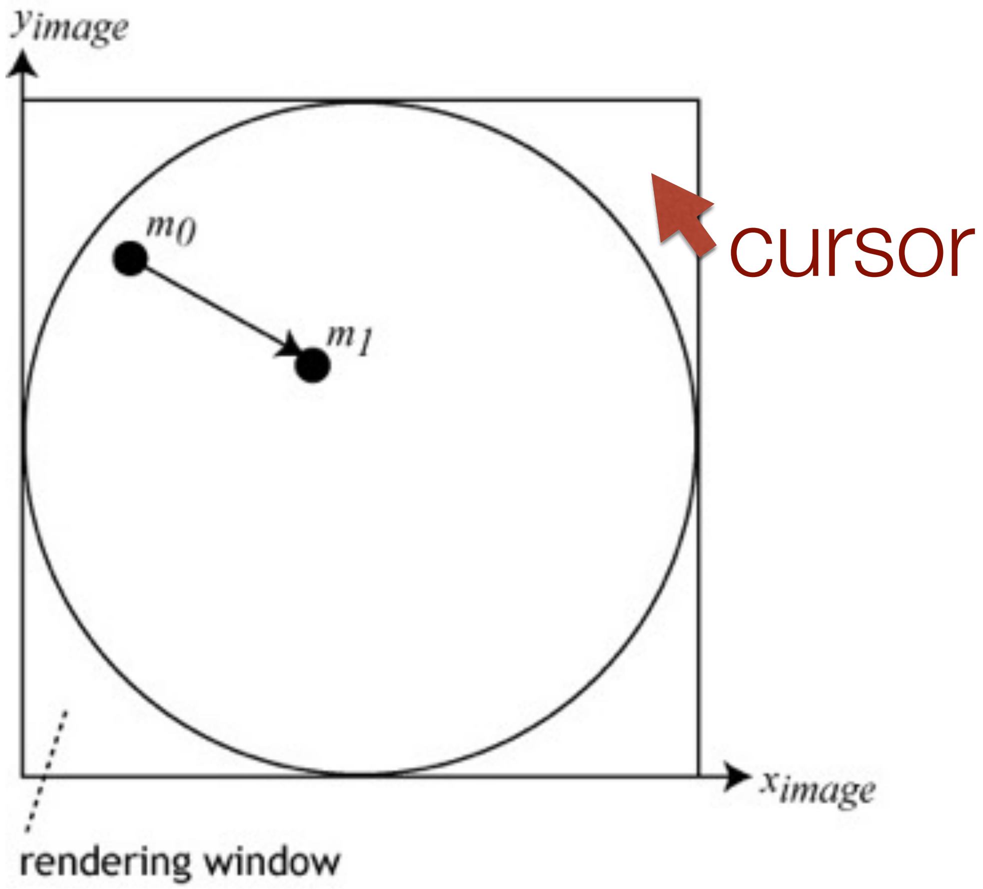
m_0	the screen space position where the mouse is pressed down
m_1	the screen space position where the mouse button is dragged to
v	m_0 mapped onto virtual trackball
w	m_1 mapped onto virtual trackball
a	rotation axis, v x w (rotation magnitude ~ to angle between v and w)

2. Trackball

- What if mouse cursor is outside sphere projection?
  - Use hypersheet instead
    https://www.opengl.org/wiki/Object_Mouse_Trackball



$$z(x', y') = \frac{r^2/2}{\sqrt{x'^2 + y'^2}}$$



2. Trackball

- https://www.opengl.org/wiki/Object_Mouse_Trackball
- 1.5 points
- `project_onto_surface`
  (`TODO 2` in `trackball.h`)
- Create rotation matrix
  (`drag TODO 3` in `trackball.h`)
- Apply the rotation
  (`TODO 3` twice in `main.cpp`)

2. Zooming

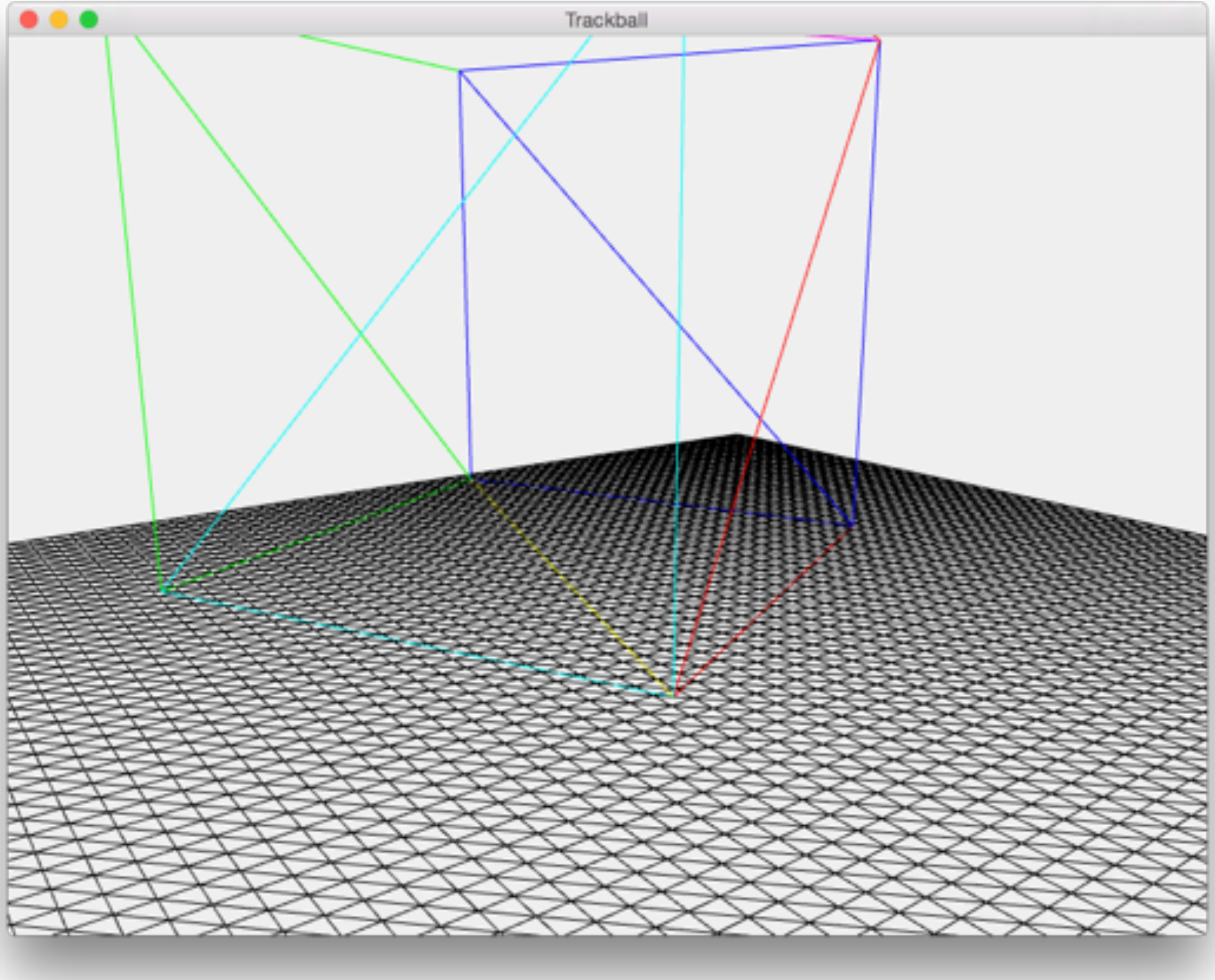
- Zoom in/out when moving the mouse cursor down/up while the right mouse button is pressed

- `TODO 4` in `main.cpp`

- 0.5 points

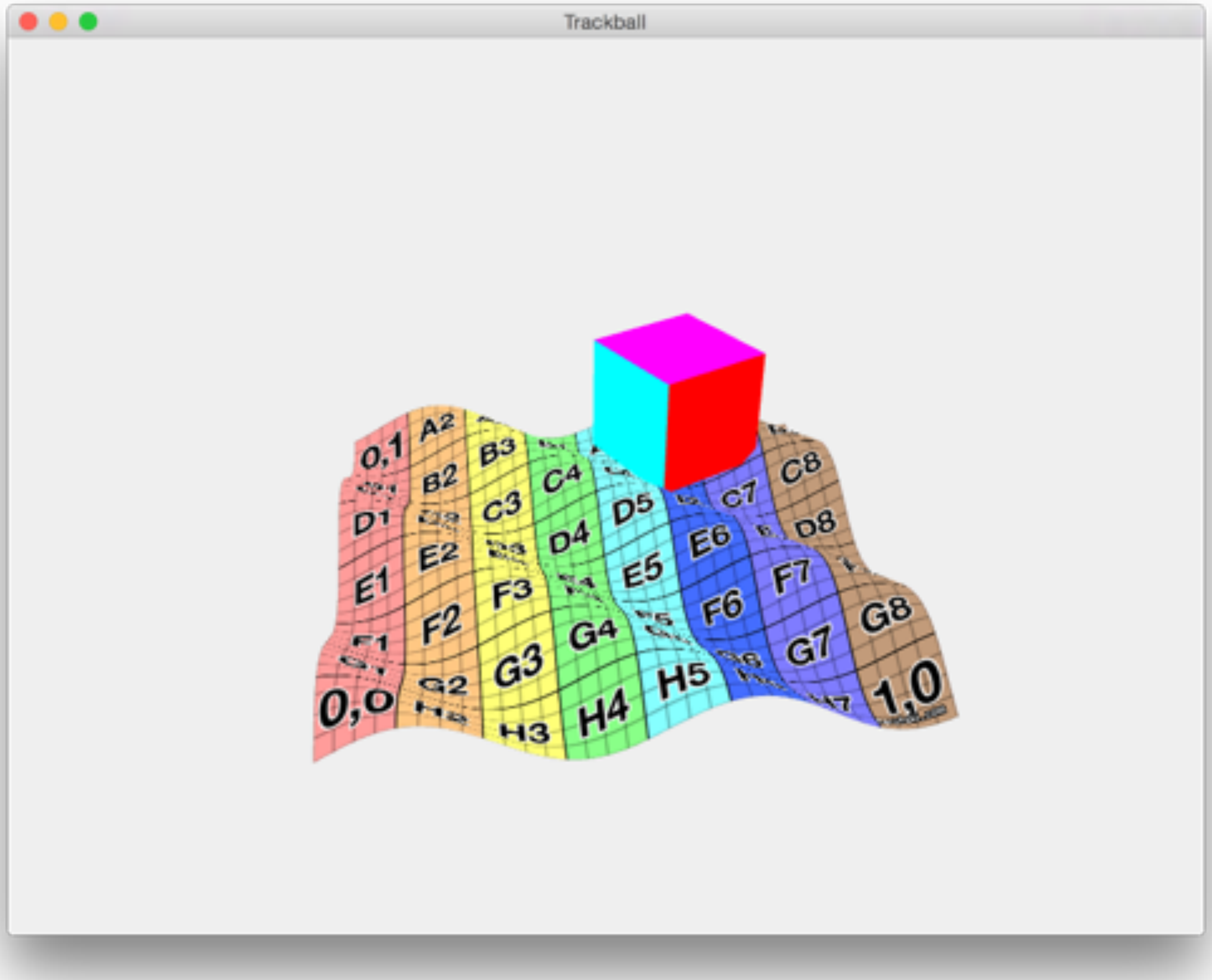
3. Triangle Grid

- Change the horizontal quad into a grid of 100x100 vertices and triangulate it
  - Fill vertex and index buffers
  - Wireframe rendering can help with debugging `glPolygonMode(…)`
  - `TODO 5` twice in `_grid/grid.h`
  - 1.5 points



3. Triangle Grid

- Animate height of vertices in grid as a sine function of time
  - `TODO 6` in `_grid/grid_vshader.glsl`
  - 0.5 points
  - 1 bonus point if you animate the grid to look like water (max 6 points possible)


Trackball
0,1
1,0
0,0