Computer Graphics

Minho Kim

Dept. of Computer Science

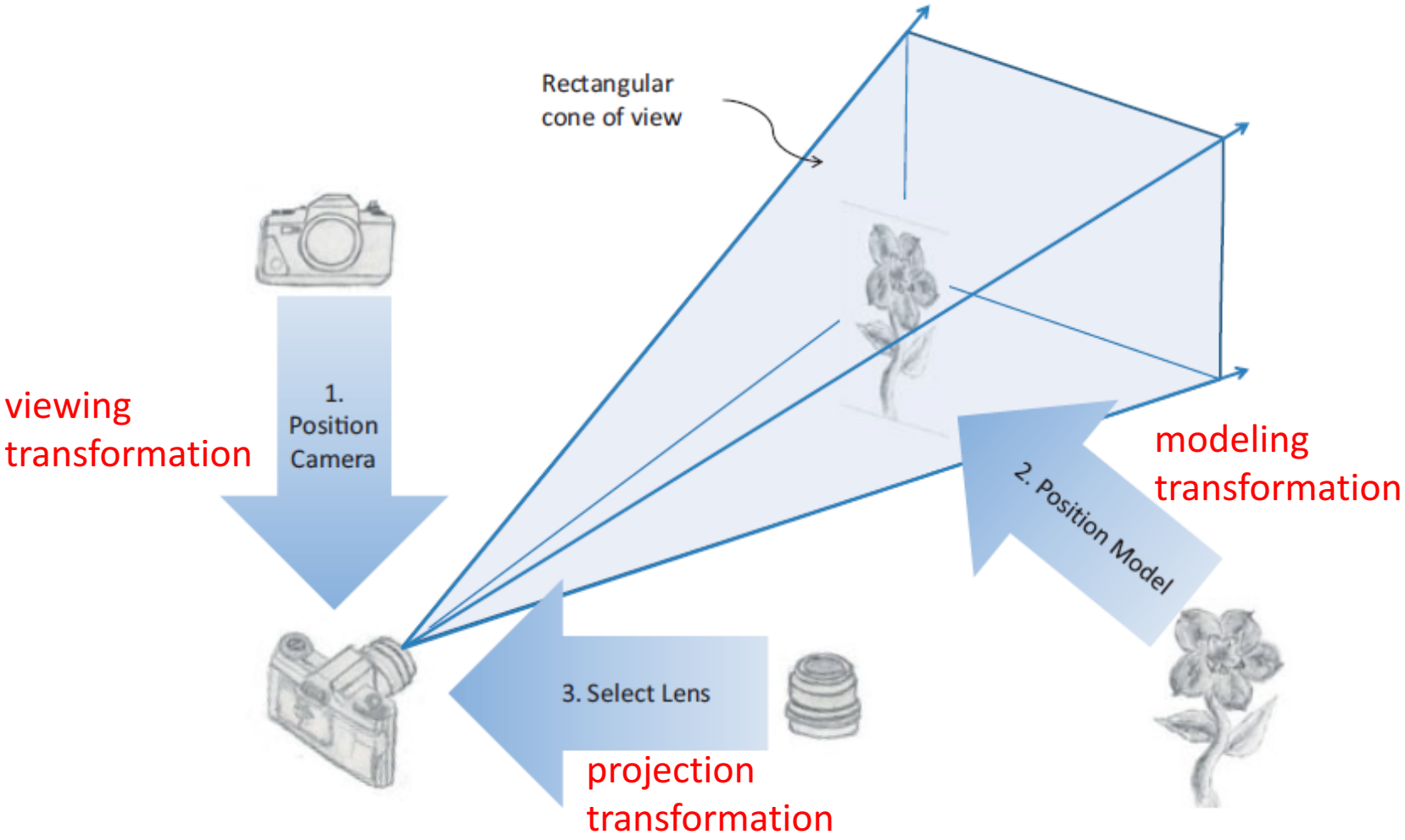
University of Seoul

Chapter 7: Toward the 3D World

What to Learn

- Representing the user's view into the 3D world
  – camera transformation (position and direction)
- Controlling the volume of 3D space that is viewed
  – setting camera lens
- Clipping
- Handling foreground and background objects
  – hidden surface removal
- Drawing a 3D object (a cube)

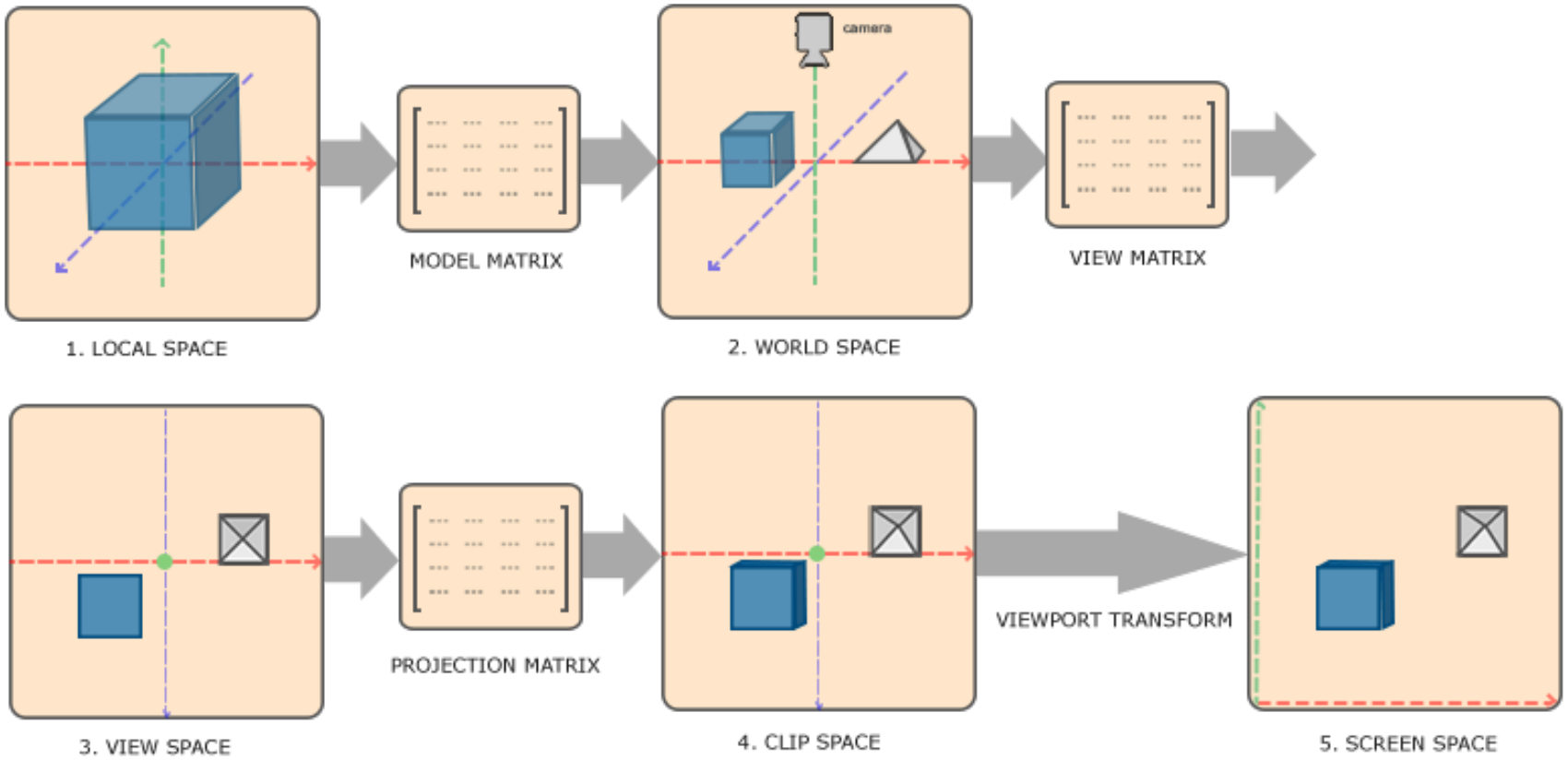
Transformations – Photography Analogy



OpenGL Transformation

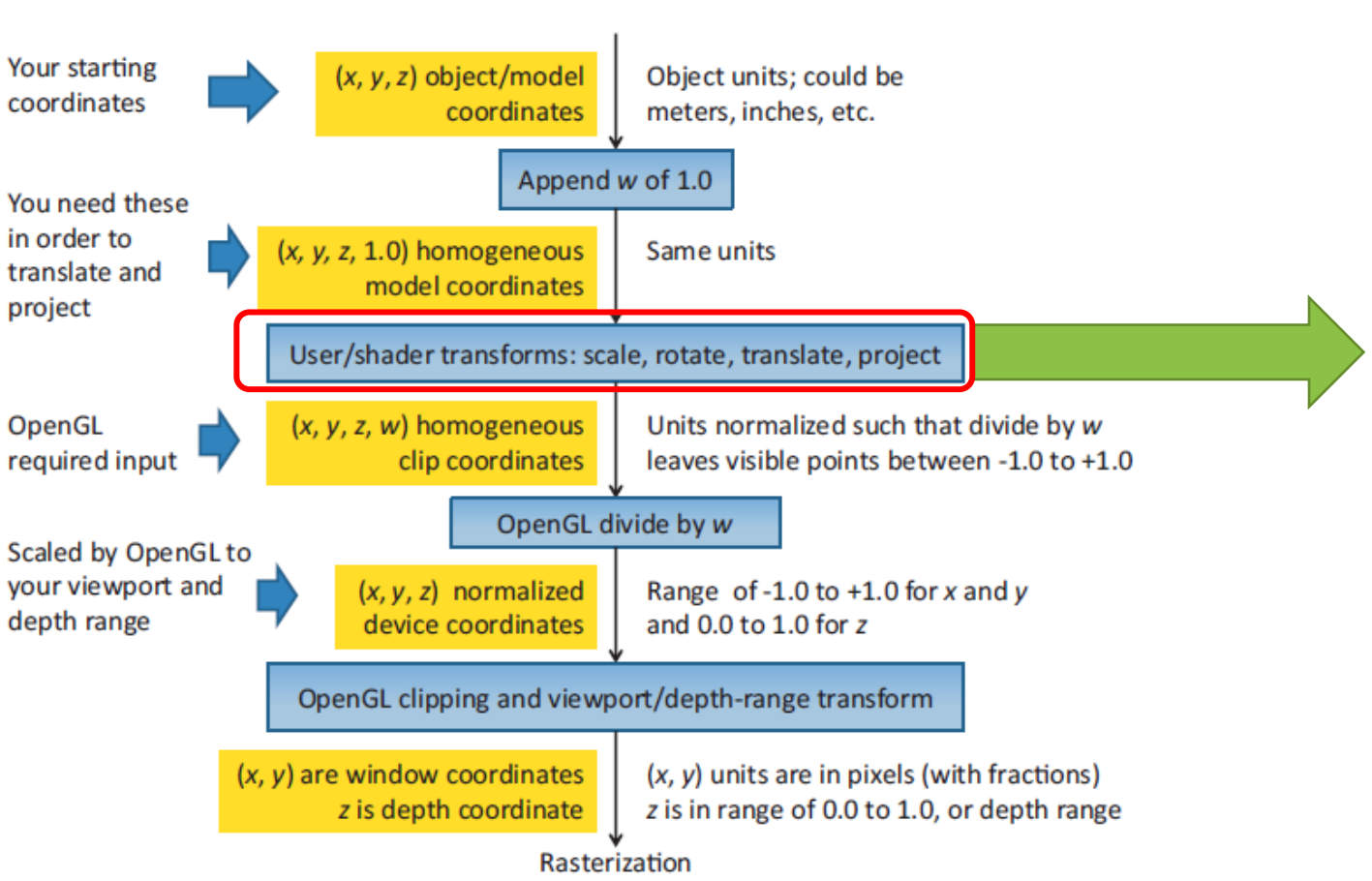
- Model-view transformation
  - We can fix the camera and transform the objects only
    → viewing & modeling transformations can be combined into one
    → **model-view transformation**
  - Unified space for all the objects are assembled into one the scene to view → "**eye space**" or "**camera space**"
- We are responsible to apply the three transformations (model/viewing/projection) to the vertices in the vertex shader
  → The output vertex position of the vert shader should have been applied all the three transformations and represented in "**clip coordinates**"
- We are also responsible to do the "**viewport transformation**"

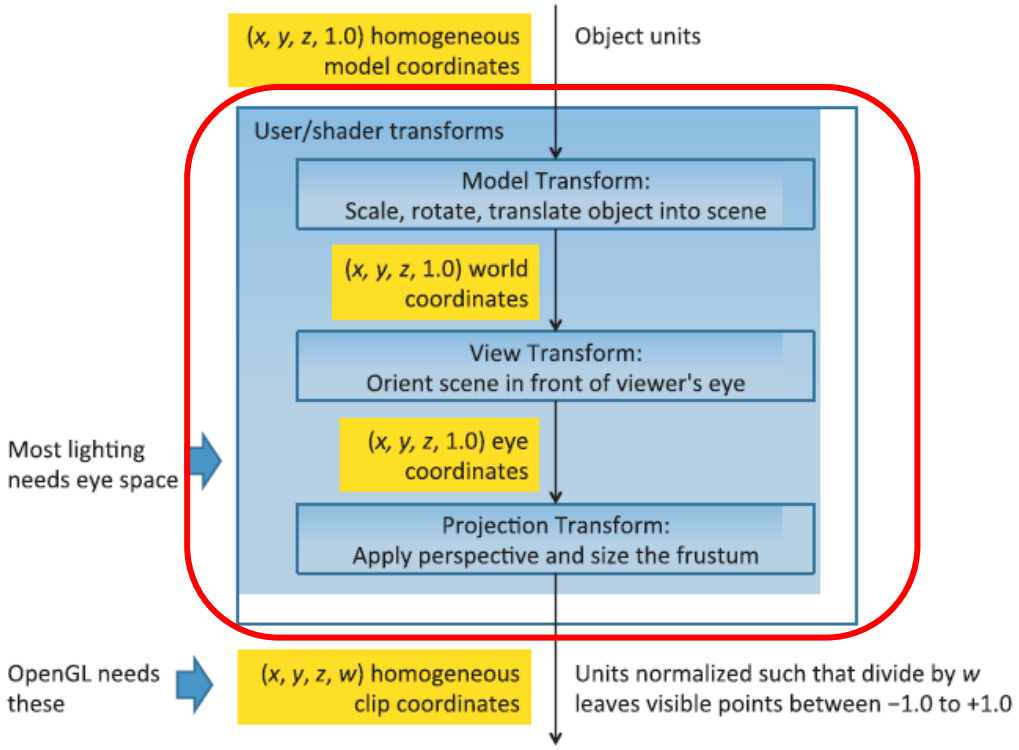
OpenGL Coordinate Systems



(image courtesy of learnopengl.com)

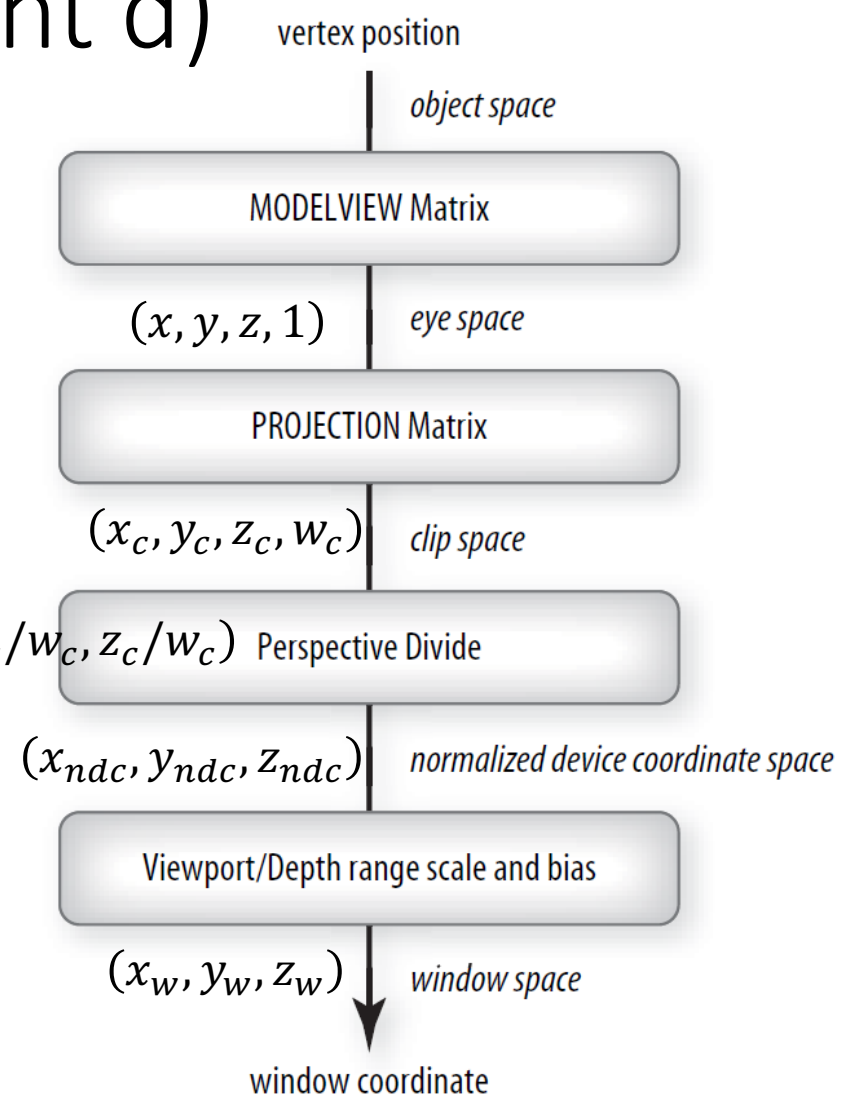
OpenGL Coordinate Systems (cont'd)





Classical Coordinate Systems (cont'd)

- Object space → (World space) → Eye space → Clip space → NDC space → Window space
- Usually, a vertex shader needs to generate `gl_Position` in clip space by multiplying an MVP (Model-View-Projection) matrix
- After the vertex shader, primitives are clipped in the clip space against the view volume
  - $-w_c \leq x_c \leq w_c, -w_c \leq y_c \leq w_c, -w_c \leq z_c \leq w_c$
- After clipping, "perspective division" is applied so that the coordinates are converted to be in the NDC (normalized device coordinates)
  - $(x_c, y_c, z_c, w_c) \rightarrow (x_c/w_c, y_c/w_c, z_c/w_c)$
- Note that the z-axis is flipped (away from the viewpoint) from the clip space

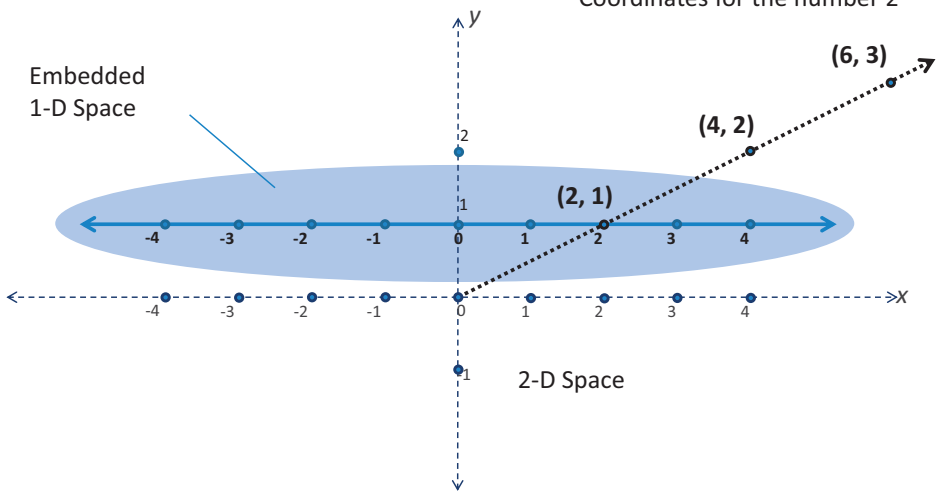


(orange book)

Model-View Transformation

- Model → View(camera transformation) → Projection(lens transformation)
- Model & view transformations are combined into one → Model-View transformations
- The result (`gl_Position`) should be in the clip coordinates
- Represented as homogeneous coordinates
- Represented by 4x4 matrices → linear transformations
  - Ex) scaling, translation, rotation, etc.
  - Composite transformations are not commutative!
- **Usual model transformations**
  - **re-orienting** around its own origin and then **positioning** itself in the world coordinates
  - scaling (mostly optional) → rotations (re-orientation) → translation (positioning)
- **Usual view (camera) transformations** (**applied to the world**)
  - rotation around y-axis (tracking the camera around the origin on the world x-y plane)
    → rotation around x-axis (tilting the camera)
    → translation along z-axis (moving the camera away from the origin of the world)

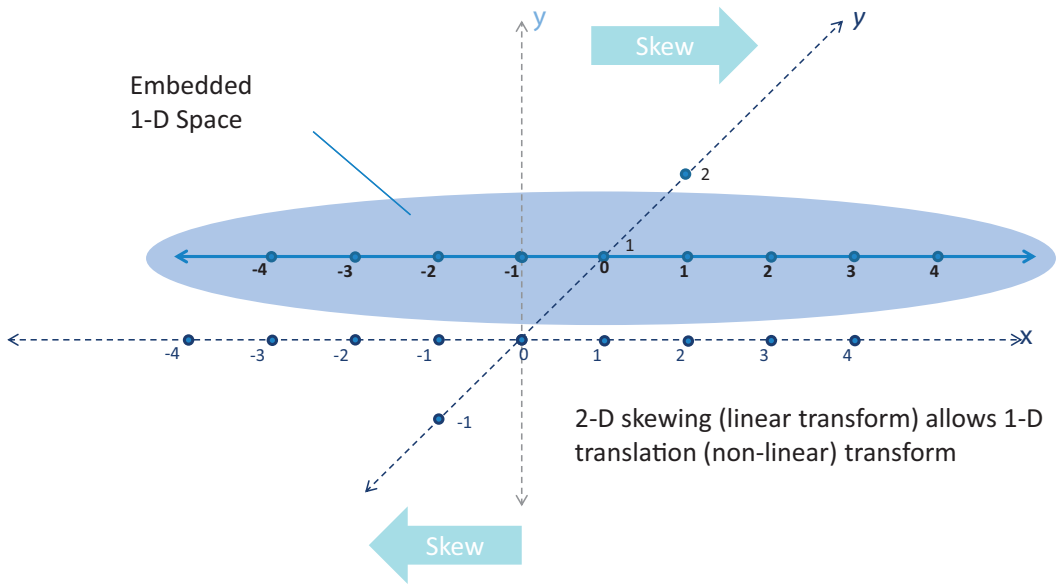
Homogeneous Coordinates



- Two advantages
  - Perspective viewing is possible
  - Translation (one of affine transformations) can be represented as a linear transformation
- Additional fourth component (w)
  - $w = 1$: points
  - $w = 0$: point at infinity → vectors
- Two coordinates are the same if one can be obtained by scaling the other → "directions"
  - Ex) $(2,3,5,1) = (4,6,10,2)$

Homogeneous Coordinates (cont'd)

- 3D translation can be achieved by a 4D shear transformation
  - $$\begin{bmatrix} 1 & 0 & 0 & \alpha \\ 0 & 1 & 0 & \beta \\ 0 & 0 & 1 & \gamma \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x + \alpha \\ y + \beta \\ z + \gamma \\ 1 \end{bmatrix}$$
  - Vectors ($w = 0$) are not affected



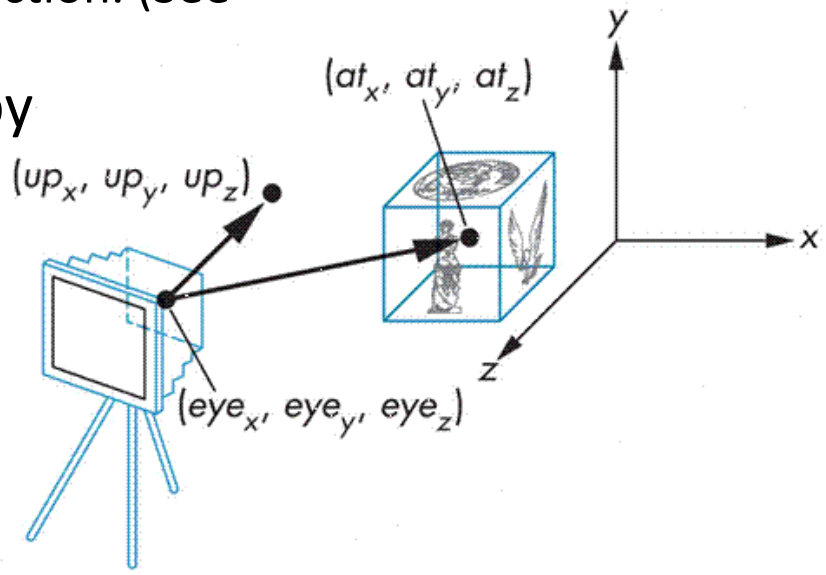
- Perspective projection can be achieved by "division by w"
  - Large w (far from the camera) → (x, y, z) scaled down

Camera Setting

- Camera transformation
  - Where you are looking them, and at which part of the scene are you looking?
  - Position + viewing direction
- Camera lens
  - Given the viewing direction, where can you actually see?
  - Orthogonal/perspective
  - Minimum/maximum viewing distance

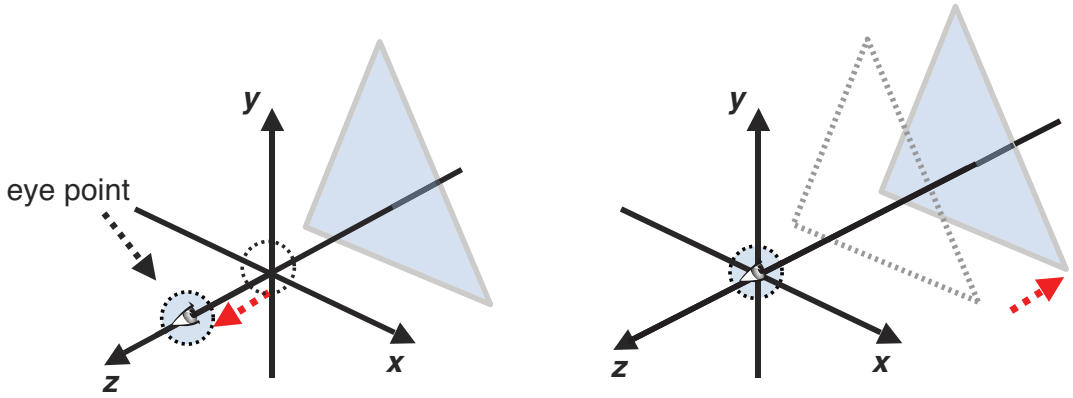
Eye Point, Look-At Point, and Up Direction

- Which information do we need to specify a camera in the 3D space?
- Can be uniquely specified by (1) eye point, (2) look-at point, and (3) up direction.
  - Look-at point & up direction are not unique for a given setting.
  - What if one of the information is missing?
  - What if the up direction is not orthogonal to the look-at direction? → The up direction is projected to the plane orthogonal to the look-at direction. (See `Matrix4.setLookAt()`.)
- Defined by a matrix called view matrix which can be set by `Matrix4.setLookAt()`.
- Default values
  - eye point: (0,0,0)
  - look-at point: (0,0,-1)
  - up direction: (0,1,0)



Viewing Transformation

- There is no special transformation for the camera. The camera is ALWAYS (1) located at (0,0,0), (2) pointing at the –z direction, and (3) its up direction is +y direction.
- Applying the viewing transformation, we are in fact transforming all the objects in the scene with respect to the camera, so that it “looks like” we are transforming the camera.
- Look into `Matrix4.setLookAt()` source code.



Example #1: `LookAtTriangles`

Example #1: `LookAtTriangles`

- http://rodger.global-linguist.com/webgl/ch07/LookAtTriangles.html
- Three triangles are rendered near the origin.
  - With their faces parallel to the xy plane
  - With their z coordinate -0.4, -0.2, and 0.0, respectively,
- The "camera" is located at $(0.20,0.25,0.25)$ pointing at $(0,0,0)$.
- Note the drawing order of triangles. What if we re-order them?
- What to learn
  - How to set the view matrix
  - How to apply the viewing transformation in the vertex shader

Example #2: `LookAtRotatedTriangles`

Example #2: `LookAtRotatedTriangles`

- http://rodger.global-linguist.com/webgl/ch07/LookAtRotatedTriangles.html
- What to learn
  - How to apply both view and model transformations at the same time.
- Arrange objects in the space first and then move the camera to a specific location → `vec4 gl_Position = V*M*a_Position;`
  - `V`: view matrix
  - `M`: model matrix
- It is more efficient to multiply `V` & `M` in the host and pass the resulting matrix ("model view matrix") to the vertex shader. (Why?)

Lab Activities

- Modify `LookAtRotatedTriangles` such that the multiplication of the view & model matrices is done only once in the host.
  - LookAtRotatedTriangles_mvMatrix.html
- Change the eye positions. Any problem?
- Change the drawing order of the triangles. Any problem?

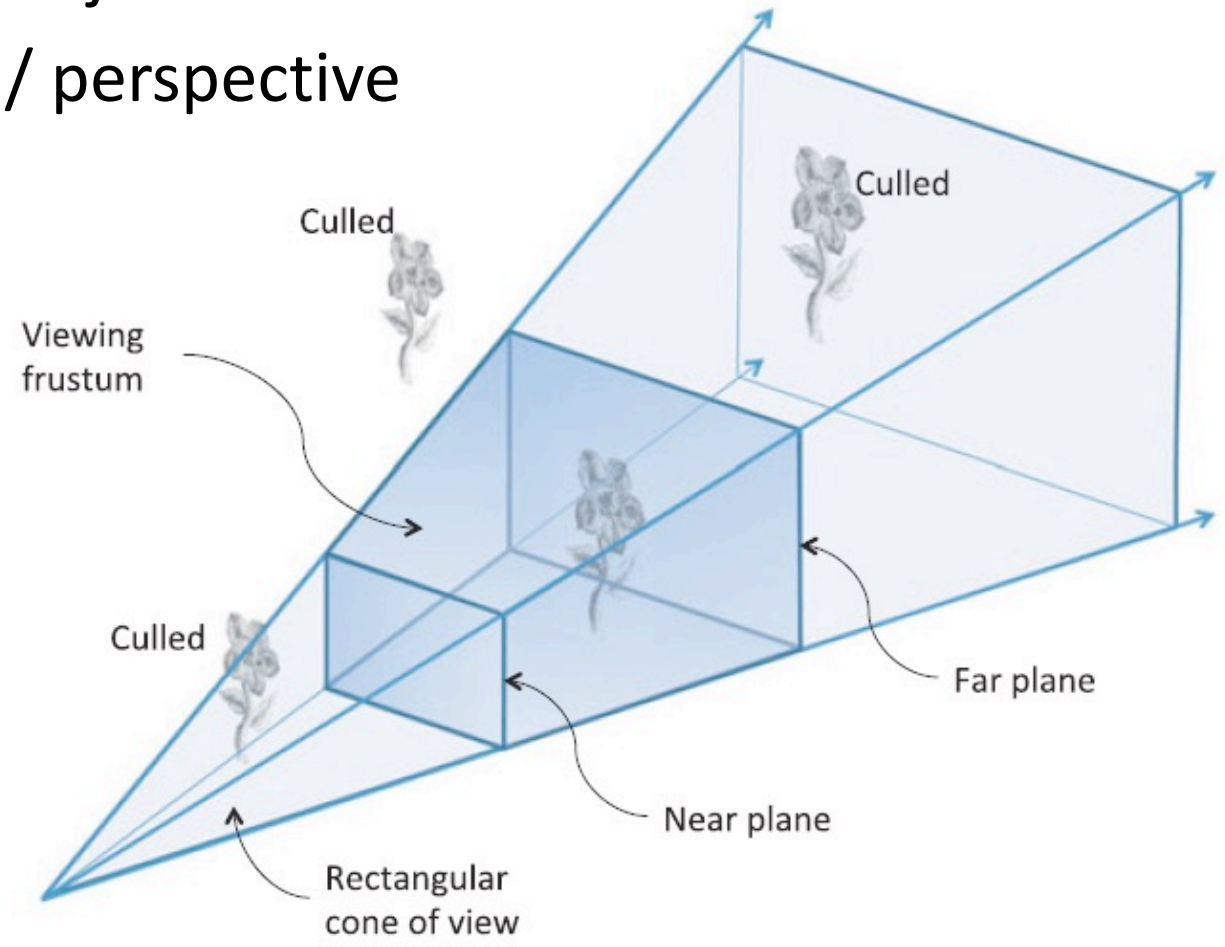
Example #3: `LookAtTrianglesWithKeys`

Example #3: `LookAtTrianglesWithKeys`

- http://rodger.global-linguist.com/webgl/ch07/LookAtTrianglesWithKeys.html
- Changes the x coordinate of the camera using the left & right arrow keys.
- What to learn
  - How to handle JavaScript keyboard events
    - keycode list can be found here.
  - To see how the “visible range” affects the rendering.

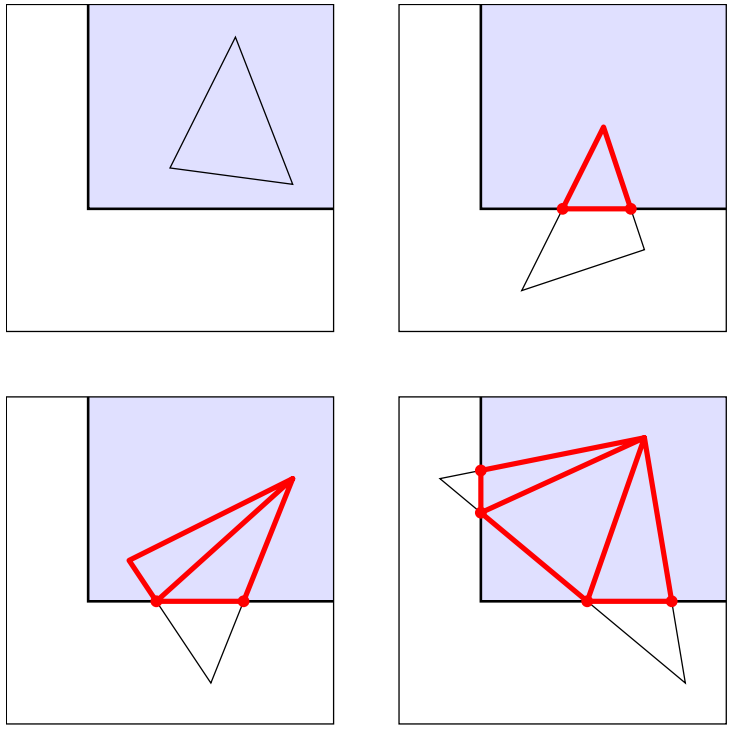
Viewing Frustum

- Defined by a projection matrix
- Orthographic / perspective



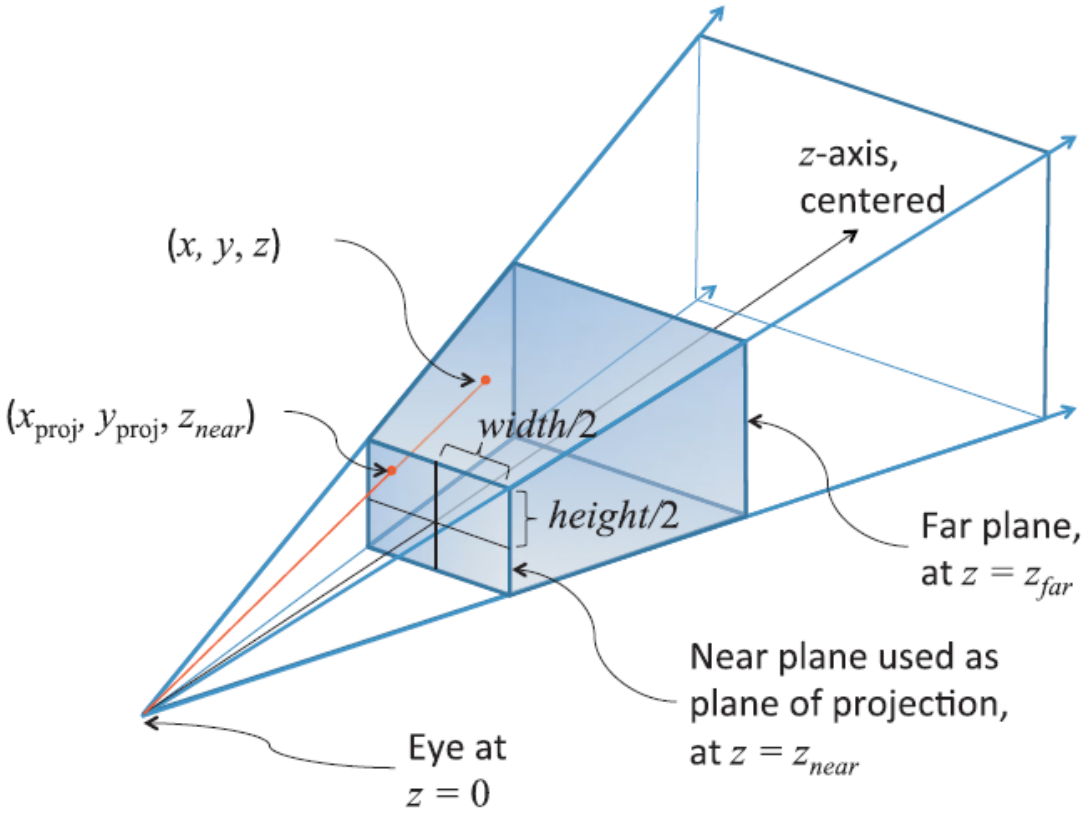
Frustum Clipping

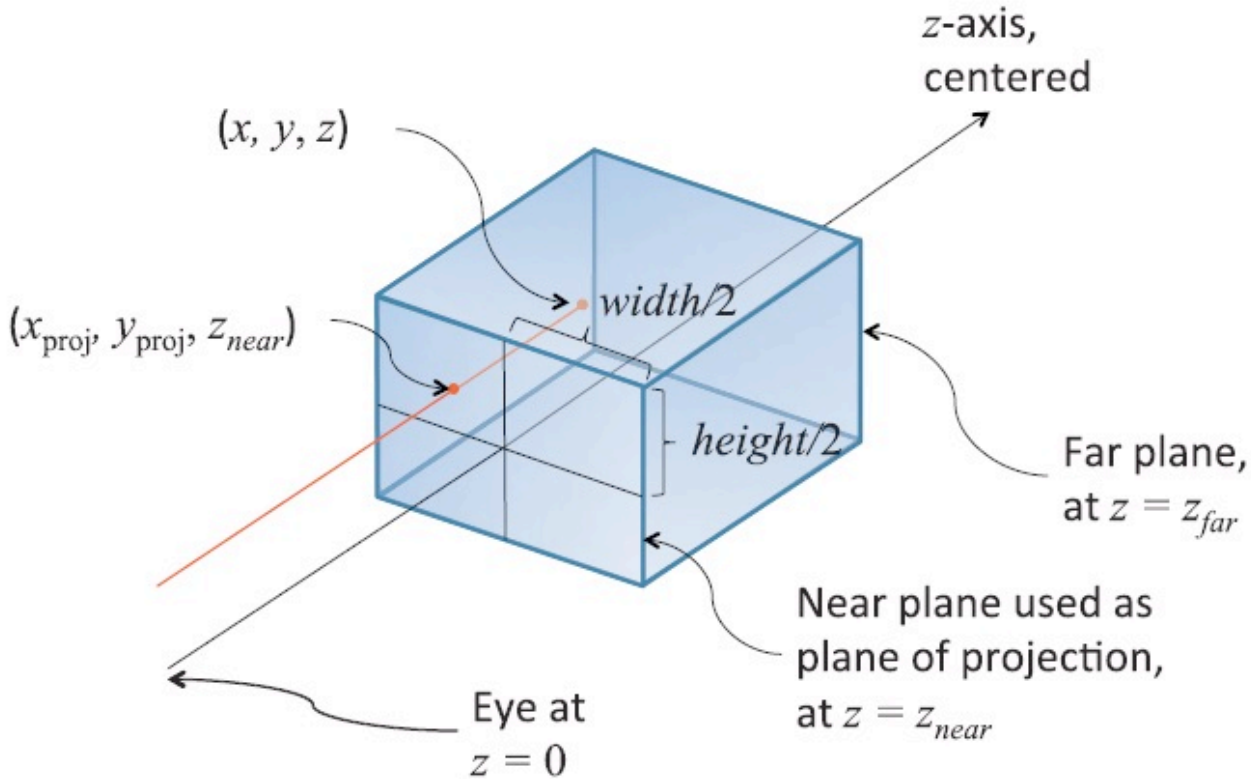
- Computed in clip space
- May generate more primitives



(Jason L. McKesson)

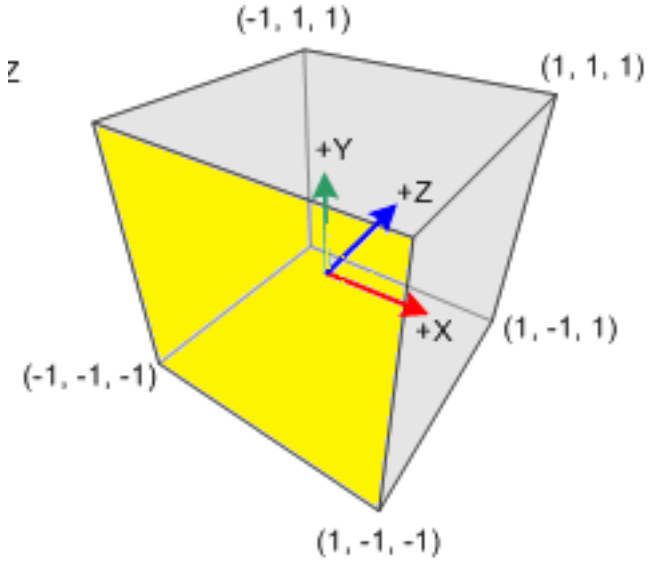
Viewing Models – Perspective & Orthographic





Projection Matrices

- Input: homogeneous coordinates $(x, y, z, 1)$ in the eye space
- Output: coordinates in the clip space (x_c, y_c, z_c, w_c)
  - w_c components may not be 1 anymore
- The view frustum is mapped to the cube (NDC)
  - defined as $[-1,1]\times[-1,1]\times[-1,1]$
  - Coordinates orientation is changed (z-axis is flipped)
- After the vertex shader, primitives are clipped in the clip space against the view volume
  - $-w_c \leq x_c \leq w_c, -w_c \leq y_c \leq w_c, -w_c \leq z_c \leq w_c$
- We only have `set*()` functions for setting projection matrices. Why?



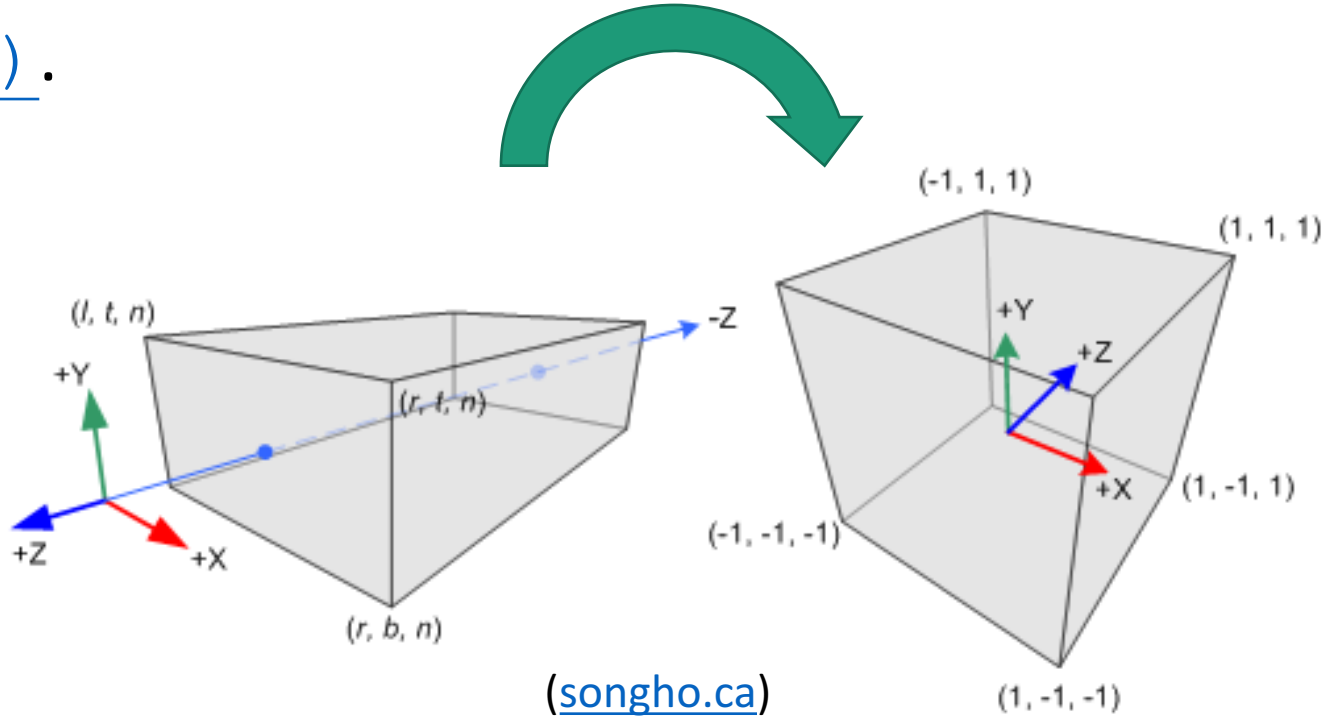
NDC (Normalized Device Coordinates) cube

Orthographic Projection

- No “depth perception” – same sizes regardless of the distance from the camera
- Defined as a **projection matrix** set by `Matrix4.setOrtho()`.
- Near/far planes can be located behind the camera – not recommended
- Easy to compute
  - How each vertex is projected to the image plane
  - How far each vertex is
- Useful for designing process (e.g. CAD / 3D modeling software)
- Orthographic projection matrix – converts from the eye (camera) coordinates to the clip coordinates

Orthographic Projection

- Set by `Matrix4.setOrtho()`.
- $$\begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{-2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
- Verification
  - $(l, t, -n, 1) \rightarrow (-1,1,-1,1)$
  - $(r, b, -f, 1) \rightarrow (1,-1,1,1)$



(songho.ca)

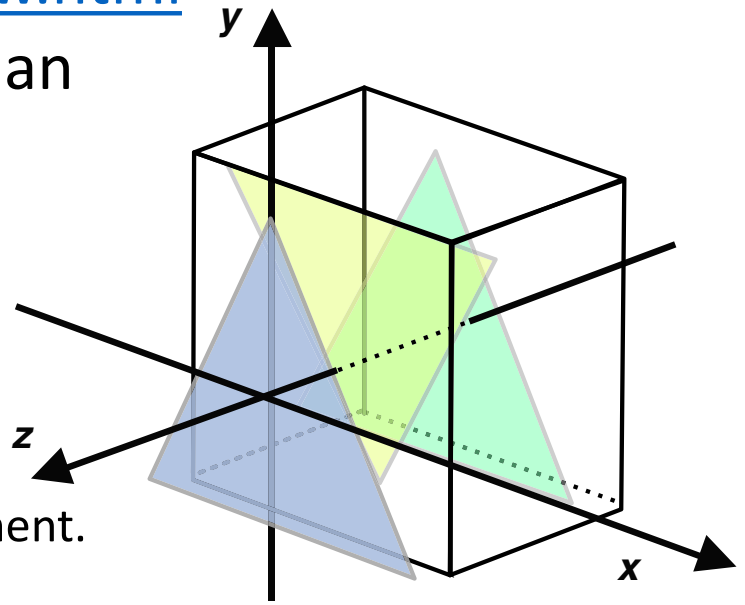
Orthographic Projection: Default Projection

- Applying no projection matrix == setting the identity matrix as the projection matrix
- Identity matrix is the orthographic matrix set by calling `Matrix4.setOrtho(-1,1,-1,1,1,-1)` where `near==1` and `far==-1`.
- Therefore, it has the same effect as flipping the z-axis. (objects with smaller z coordinates are closer to the camera.)
  → To avoid confusion, we always have to set the projection matrix.

Example #4: `OrthoView`

Example #4: `OrthoView`

- http://rodger.global-linguist.com/webgl/ch07/OrthoView.html
- Dynamically changes the near and far clipping planes of an orthographic projection matrix.
  - Left/right: near plane
  - Up/down: far plane
- What to learn
  - How to apply an orthographic projection
  - How to dynamically modify an HTML element using JavaScript
    - By assigning a string to the `innerHTML` property of an HTML element.
    - The HTML element, stored in `nf`, is passed to the event handler.
- Some value may be interpreted to a string not, e.g., an integer. To ensure it to be interpreted to an integer, use `parseInt()` function.



Example #5:

`LookAtTrianglesWithKeys_ViewVolume`

Example #5: `LookAtTrianglesWithKeys_ViewVolume`

- http://rodger.global-linguist.com/webgl/ch07/LookAtTrianglesWithKeys_ViewVolume.html
- What to learn
  - How to apply a projection matrix combined with a view transformation
  - How to zoom in/out by changing the view volume
  - To see what happens if the aspect ratio of the view volume and the `<canvas>` do not match.

Lab Activities

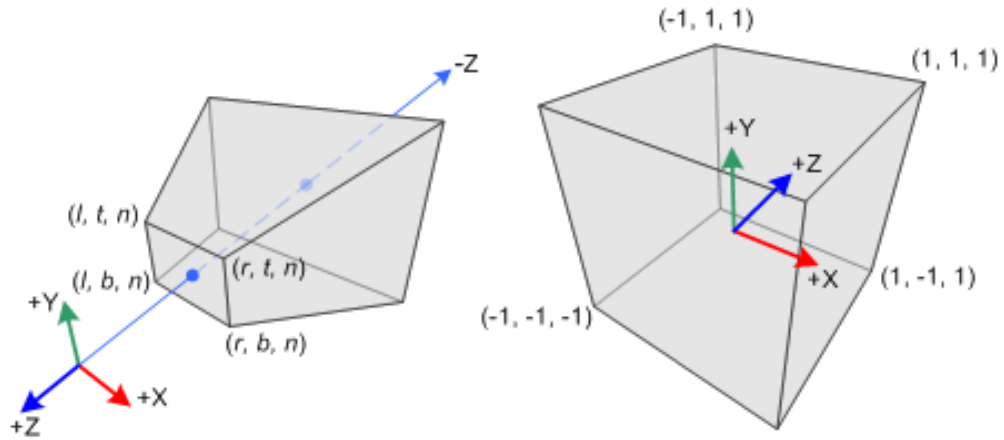
- Try changing the parameters for `setOrtho()` function and see the result.
- Try to make it look “zoomed in/out” by setting the parameters properly.
- http://rodger.global-linguist.com/webgl/ch07/OrthoView_halfSize.html
- http://rodger.global-linguist.com/webgl/ch07/OrthoView_halfWidth.html

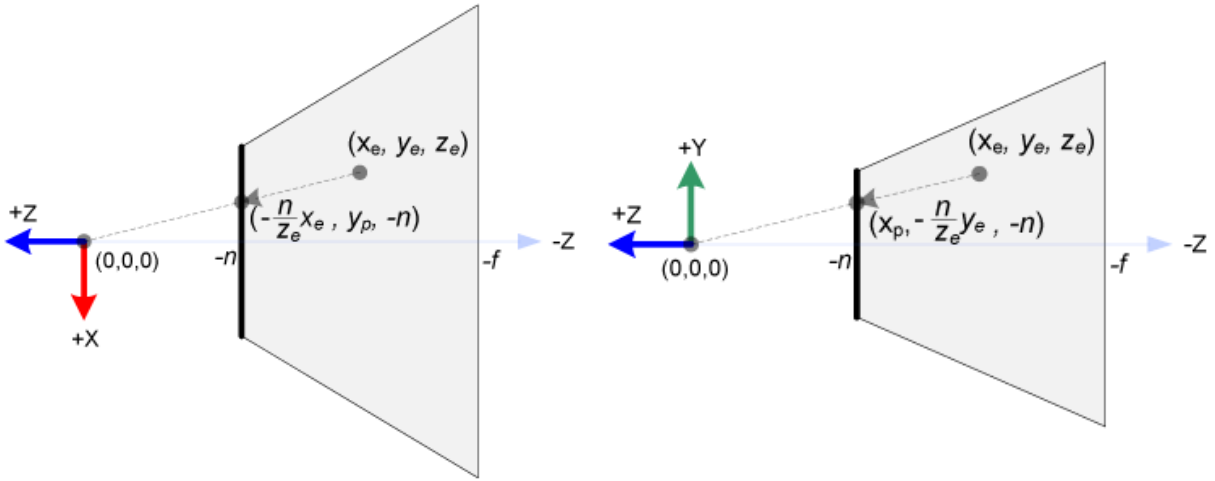
Perspective Projection

- Depth perception
- Near/far planes should be always in front of the camera
- Near/far parameters are set to positive values, but they are located on the negative z axis.
- The perspective projection matrix converts the eye (camera) coordinates to the clip coordinates
- While we can directly compute how each vertex is projected onto the image plane, we first convert the view frustum to the box-shaped clip space (NDC, Normalized Device Coordinates) and then apply the orthographic projection. → Easier clipping
- Reading material: https://paroj.github.io/gltut/Positioning/Tut04%20Perspective%20Projection.html

Perspective Projection `setFrustum()`

- $\begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$
- Verification
  - $(x, y, -n, 1) \rightarrow \left(\frac{n(2x-r-l)}{r-l}, \frac{n(2y-t-b)}{t-b}, -n, n\right)$
  - $(x, y, -f, 1) \rightarrow \left(\frac{2xn-f(r+l)}{r-l}, \frac{2yn-f(t+b)}{t-b}, f, f\right)$
  - $(r, t, -n, 1) \rightarrow (n, n, -n, n)$
  - $\left(\frac{rf}{n}, \frac{tf}{n}, -f, 1\right) \rightarrow (f, f, f, f)$

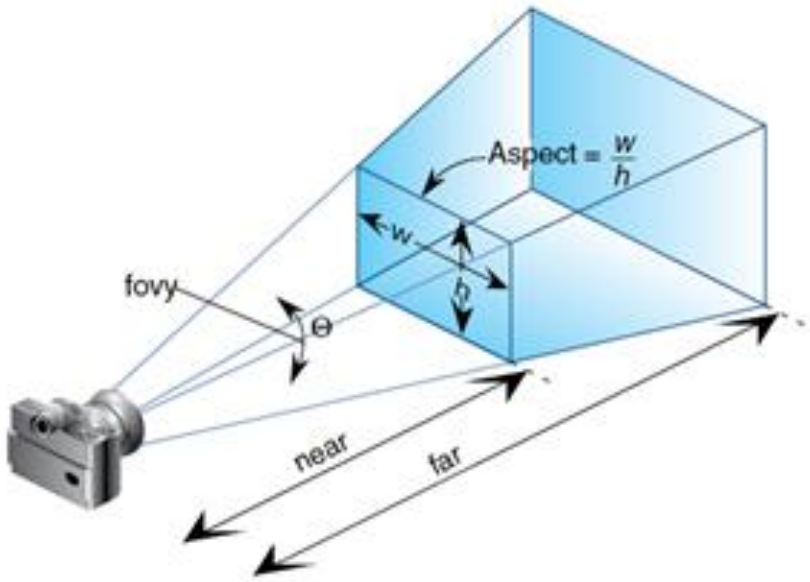




(songho.ca)

Perspective Projection `setPerspective()`

- $$\begin{bmatrix} \frac{\cot\frac{fovy}{2}}{aspect} & 0 & 0 & 0 \\ 0 & \cot\frac{fovy}{2} & 0 & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$



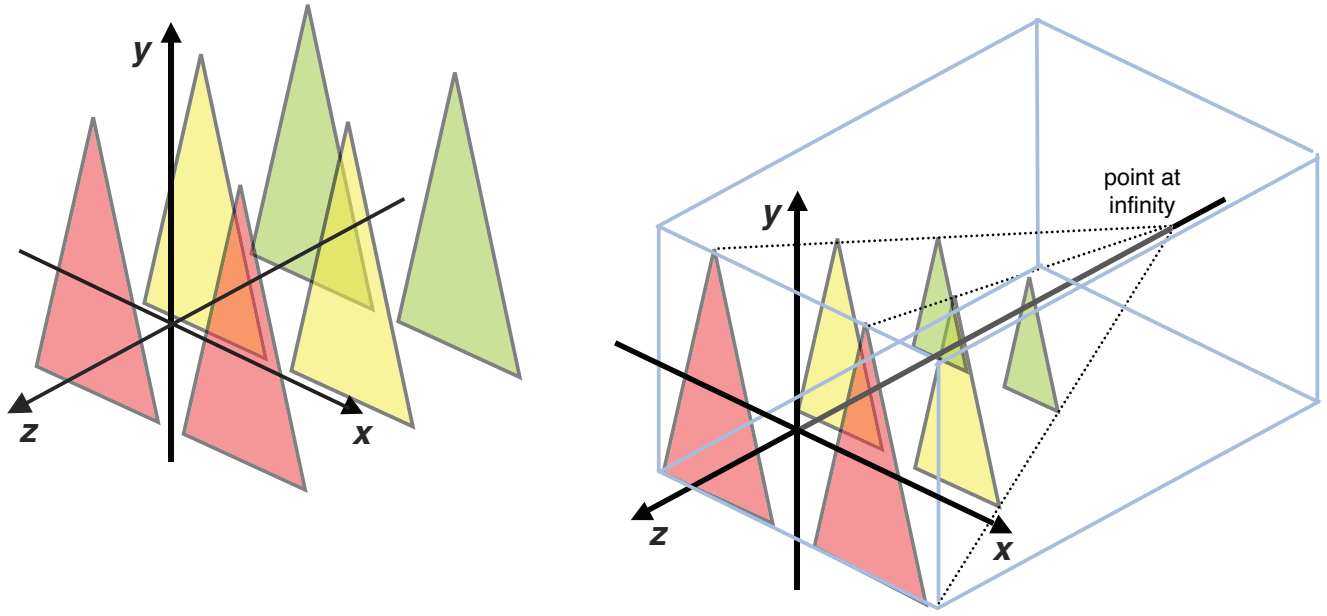
(codersource.net)

- Special case of `setFrustum()` with
  - $l = -r$
  - $b = -t$
- Alternative parameters are $fovy$ and $aspect$
- Easier to handle zoom in/out by modifying $fovy$ parameter only.

Example #5: `PerspectiveView`

Example #5: `PerspectiveView`

- http://rodger.global-linguist.com/webgl/ch07/PerspectiveView.html
- What to learn
  - How to set a perspective projection using `setPerspective()` function.



Lab Activities

- Try modifying the parameters of `setPerspective()` and see the results.

Example #6: `PerspectiveView_mvp`

Example #6: `PerspectiveView_mvp`

- http://rodger.global-linguist.com/webgl/ch07/PerspectiveView_mvp.html
- What to learn
  - How to combine projection, viewing, and modeling transformations.
- You should be careful not to mess up the transformations!
- Three matrices are multiplied in the vertex shader.
  → inefficient
- A group of three triangles are drawn twice, with different (model) transformations applied.
  - Low memory footprint, but slower performance (calling `drawArrays()` twice)
  - Better strategy: (WebGL2) instancing using `drawArraysInstanced()` or `drawElementsInstanced()`

Lab Activities

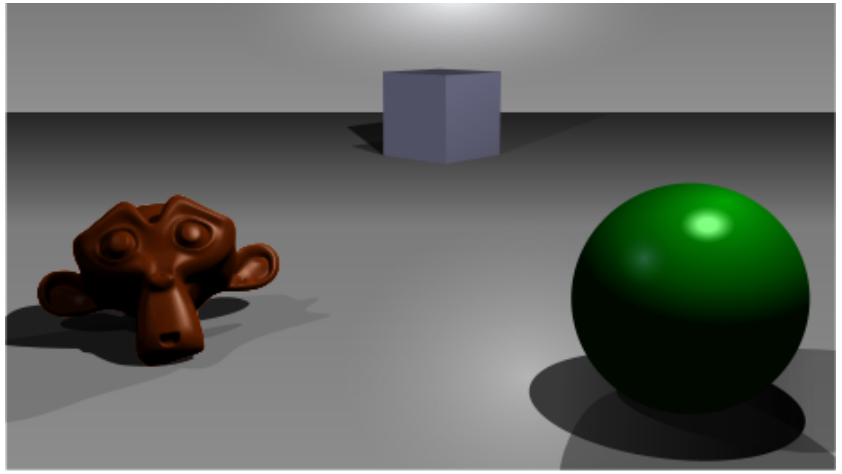
- Try modifying the code such that the matrix multiplication in the vertex shader is done at the host-side.
  - http://rodger.global-linguist.com/webgl/ch07/PerspectiveView_mvpMatrix.html
- Try modifying the code such that only two `Matrix4` objects are used.

Notes on Viewing Transformations

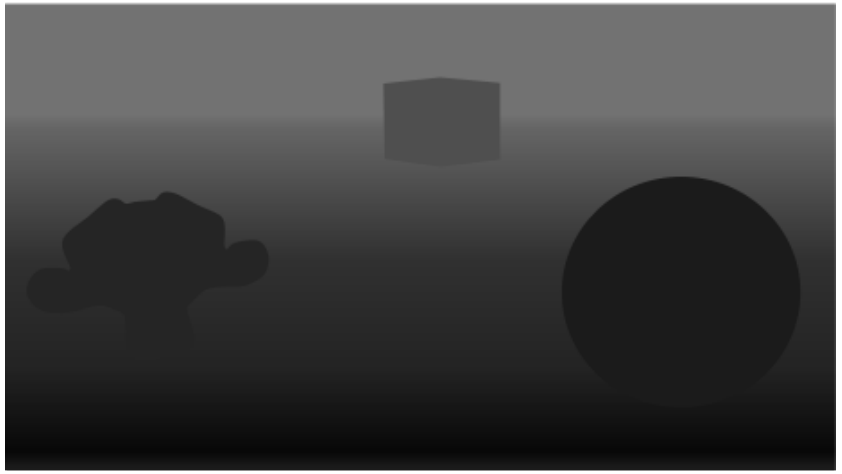
- You should apply a **translation** (which **moves the camera away from the origin**, usually to the midpoint of near & far distances) after all the model-view transformation (i.e., multiplied first) so that the objects are inside the view frustum. (`setLookAt()` function includes a translation inside) – You may not need to do that if you're using an orthographic projection including the origin, but that's not a good idea. **Please use near & far planes both located in front of the camera**.
- Check if the angles are in **radians** or **degrees**!

Hidden Surface Removal

- Hidden-surface removal using the depth buffer (z-buffer)
- Proposed by Edwin Catmull
- Each fragment's depth is compared with the existing one then overwrites it or is discarded
- Done after the fragment shader
- Comparison function set by `depthFunc()`
- Two things to be done
  - To enable depth testing at the beginning by calling `enable(gl.DEPTH_TEST)`
  - To clear the depth buffer (as well as the color buffer) everytime drawing something by calling `clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT)`



A simple three-dimensional scene



Z-buffer representation

(Wikipedia)

Example #7: `DepthBuffer`

Example #7: `DepthBuffer`

- http://rodger.global-linguist.com/webgl/ch07/DepthBuffer.html
- What to learn
  - How to enable the “hidden surface removal” feature so that we can draw objects in an arbitrary order.
- Note: If the projection matrix is not set, (identity matrix as the projection matrix) the depth values are interpreted in the opposite way.

Example #8: `Zfighting`

Example #8: `Zfighting`

- http://rodger.global-linguist.com/webgl/ch07/Zfighting.html
- Z fighting
  - If two fragments at the same location have the same (or very similar) depth values, one of them is chosen arbitrarily, which results in a “stitched artifact.”
  - Happens when drawing a triangle with its edges.
- What to learn
  - How to mitigate the z-fighting artifact using the feature “polygon offset.”
- Polygon offset
  - An offset value is computed and added to the depth
  - Needs to be enabled by `enable(gl.POLYGON_OFFSET_FILL)`
  - The offset needs to be set by `polygonOffset()`
    - `polygonOffset(1, 1)` works fine mostly.

Lab Activities

- Try commenting out `enable(gl.POLYGON_OFFSET_FILL)` and see what happens.

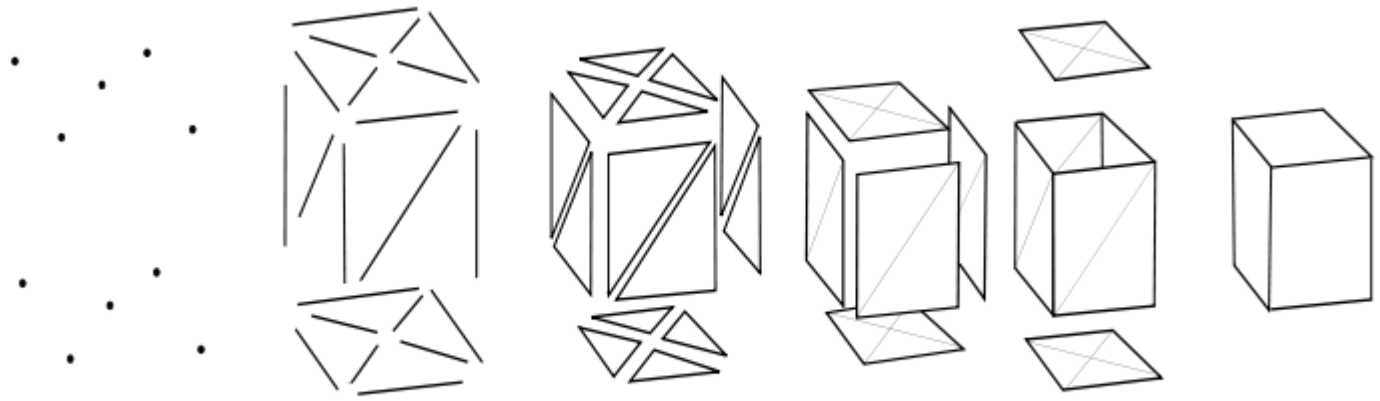
Drawing a Cube

- `gl.TRIANGLES`
  - 6x6=36 vertices, one `drawArrays()` call (https://xregy.github.io/webgl/src/cube_TRIANGLES.html)
- `gl.TRIANGLE_FAN`
  - 4x6=24 vertices, 6 `drawArrays()` calls (https://xregy.github.io/webgl/src/cube_TRIANGLE_FAN_1.html)
  - 5x4=20 vertices, 4 `drawArrays()` calls (https://xregy.github.io/webgl/src/cube_TRIANGLE_FAN_2.html)
- `gl.TRIANGLE_STRIP`
  - 14 vertices, 1 `drawArrays()` call (reference) (https://xregy.github.io/webgl/src/cube_TRIANGLE_STRIP.html)

- But there are only 8 unique vertices. What is the best way?
  - Indexed rendering using `drawElements()`

Polygonal Mesh Data Structure

- How to represent a “mesh” data structure?
- Various formats with different accessibility to the **connectivity** (topology) → https://en.wikipedia.org/wiki/Polygon_mesh
- Examples: Vertex-vertex meshes, Face-vertex meshes, Winged-edge meshes, half-edge meshes

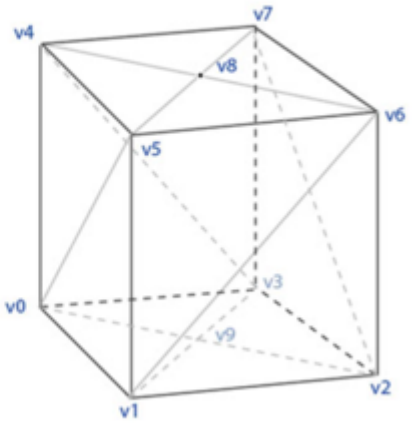


(Wikipedia)

Vertex-Vertex Meshes (VV)

Vertex List

v0	0,0,0	v1 v5 v4 v3 v9
v1	1,0,0	v2 v6 v5 v0 v9
v2	1,1,0	v3 v7 v6 v1 v9
v3	0,1,0	v2 v6 v7 v4 v9
v4	0,0,1	v5 v0 v3 v7 v8
v5	1,0,1	v6 v1 v0 v4 v8
v6	1,1,1	v7 v2 v1 v5 v8
v7	0,1,1	v4 v3 v2 v6 v8
v8	.5,.5,1	v4 v5 v6 v7
v9	.5,.5,0	v0 v1 v2 v3



(Wikipedia)

Face-Vertex Meshes

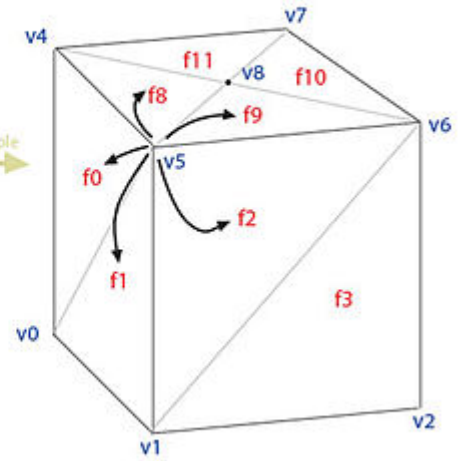
Face List

f0	v0 v4 v5
f1	v0 v5 v1
f2	v1 v5 v6
f3	v1 v6 v2
f4	v2 v6 v7
f5	v2 v7 v3
f6	v3 v7 v4
f7	v3 v4 v0
f8	v8 v5 v4
f9	v8 v6 v5
f10	v8 v7 v6
f11	v8 v4 v7
f12	v9 v5 v4
f13	v9 v6 v5
f14	v9 v7 v6
f15	v9 v4 v7

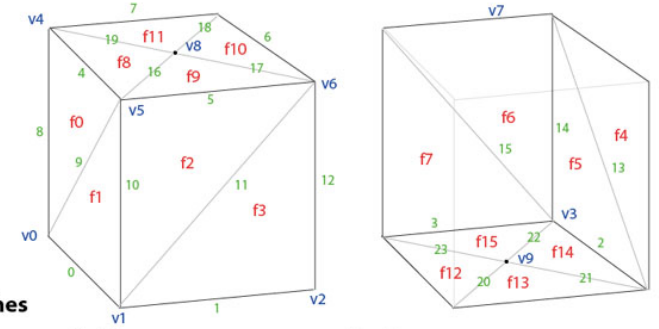
Vertex List

v0	0,0,0	f0 f1 f12 f15 f7
v1	1,0,0	f2 f3 f13 f12 f1
v2	1,1,0	f4 f5 f14 f13 f3
v3	0,1,0	f6 f7 f15 f14 f5
v4	0,0,1	f6 f7 f0 f8 f11
v5	1,0,1	f0 f1 f2 f9 f8
v6	1,1,1	f2 f3 f4 f10 f9
v7	0,1,1	f4 f5 f6 f11 f10
v8	.5,.5,0	f8 f9 f10 f11
v9	.5,.5,1	f12 13 14 15

example



(Wikipedia)



Winged-Edge Meshes

Face List

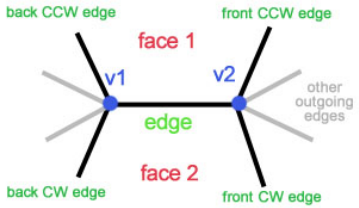
f0	4 8 9
f1	0 10 9
f2	5 10 11
f3	1 12 11
f4	6 12 13
f5	2 14 13
f6	7 14 15
f7	3 8 15
f8	4 16 19
f9	5 17 16
f10	6 18 17
f11	7 19 18
f12	0 23 20
f13	1 20 21
f14	2 21 22
f15	3 22 23

Edge List

e0	v0 v1	f1 f12	9 23 10 20
e1	v1 v2	f3 f13	11 20 12 21
e2	v2 v3	f5 f14	13 21 14 22
e3	v3 v0	f7 f15	15 22 8 23
e4	v4 v5	f0 f8	19 8 16 9
e5	v5 v6	f2 f9	16 10 17 11
e6	v6 v7	f4 f10	17 12 18 13
e7	v7 v4	f6 f11	18 14 19 15
e8	v0 v4	f7 f0	3 9 7 4
e9	v0 v5	f0 f1	8 0 4 10
e10	v1 v5	f1 f2	0 11 9 5
e11	v1 v6	f2 f3	10 1 5 12
e12	v2 v6	f3 f4	1 13 11 6
e13	v2 v7	f4 f5	12 2 6 14
e14	v3 v7	f5 f6	2 15 13 7
e15	v3 v4	f6 f7	14 3 7 15
e16	v5 v8	f8 f9	4 5 19 17
e17	v6 v8	f9 f10	5 6 16 18
e18	v7 v8	f10 f11	6 7 17 19
e19	v4 v8	f11 f8	7 4 18 16
e20	v1 v9	f12 f13	0 1 23 21
e21	v2 v9	f13 f14	1 2 20 22
e22	v3 v9	f14 f15	2 3 21 23
e23	v0 v9	f15 f12	3 0 22 20

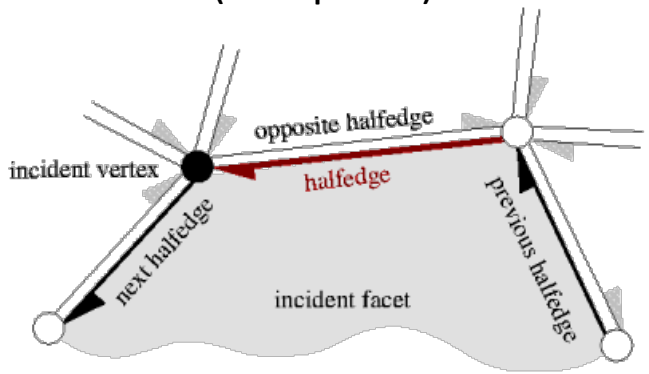
Vertex List

v0	0,0,0	8 9 0 23 3
v1	1,0,0	10 11 1 20 0
v2	1,1,0	12 13 2 21 1
v3	0,1,0	14 15 3 22 2
v4	0,0,1	8 15 7 19 4
v5	1,0,1	10 9 4 16 5
v6	1,1,1	12 11 5 17 6
v7	0,1,1	14 13 6 18 7
v8	.5,.5,0	16 17 18 19
v9	.5,.5,1	20 21 22 23



Winged Edge Structure

(Wikipedia)



(Half-edge mesh @CGAL)

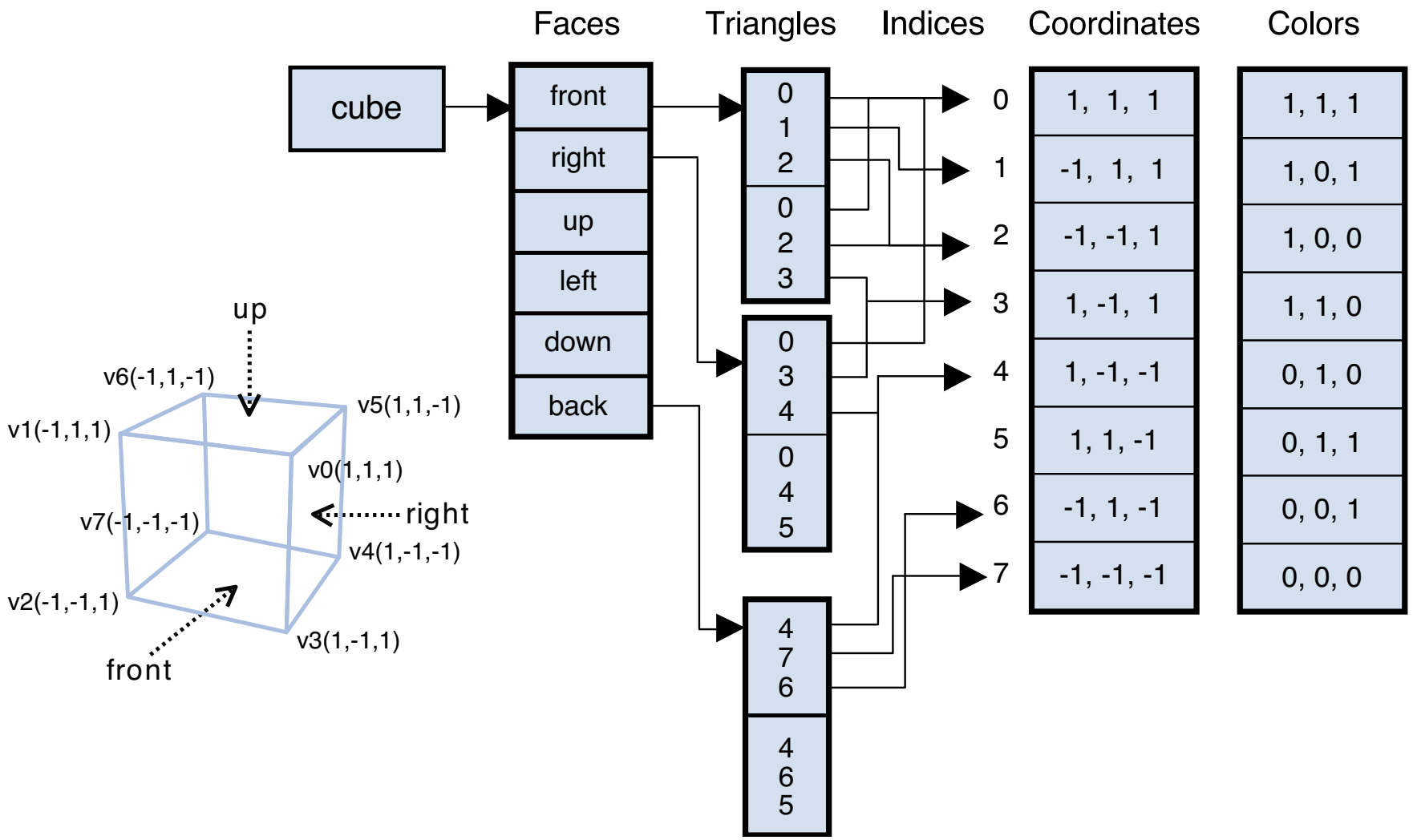
Indexed Rendering

- Drawing using two buffers by `drawElements()`.
- Vertex buffer
  - The buffer we have used so far.
  - Contains the attributes of vertices (usually float type)
  - Created by `createBuffer()` and bound by `bind(gl.ARRAY_BUFFER,*)`.
  - Vertex indices are implicitly defined sequentially.
  - Queried by `getParameter(gl.ARRAY_BUFFER_BINDING)`
- Index buffer
  - Triangles are defined indirectly by vertex indices
    → integer type (byte or short) buffer required
  - Created by `createBuffer()` and bound by `bind(gl.ELEMENT_ARRAY_BUFFER,*)`.
  - Queried by `getParameter(gl.ELEMENT_ARRAY_BUFFER_BINDING)`

Indexed Rendering (cont'd)

- The indices are usually created by a modeling tool, e.g., Blender.
  - Try exporting a model file from Blender in, e.g., wavefront `obj` format.
- Pros
  - Reduced memory to store vertex attributes
- Cons
  - Overhead handling indices

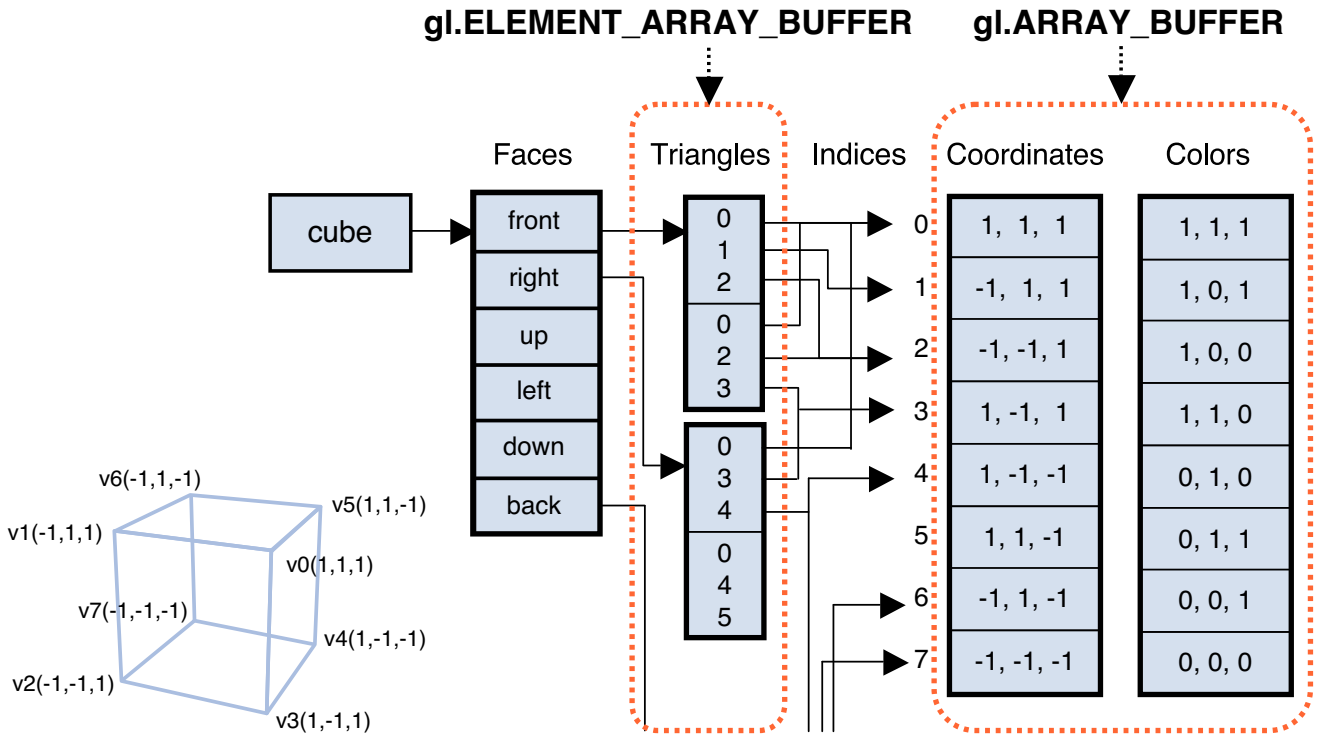
Indexed Rendering of a Cube



Example #9: `HelloCube`

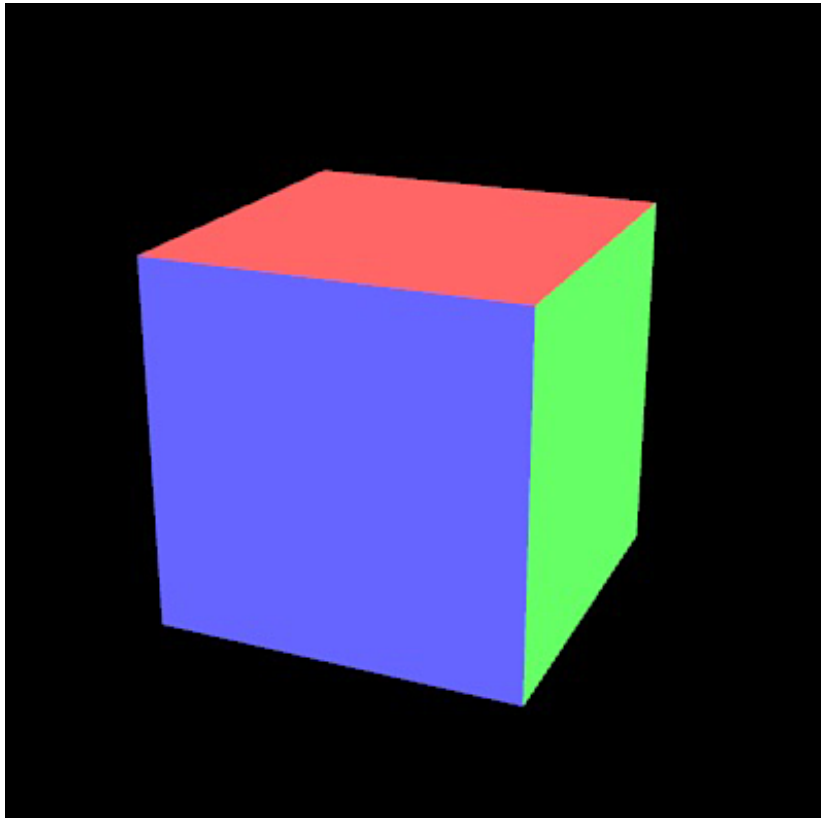
Example #9: `HelloCube`

- http://rodger.global-linguist.com/webgl/ch07/HelloCube.html
- What to learn
  - Indexed drawing using `drawElements()`



Lab Activities

- How can we assign different color to each face?

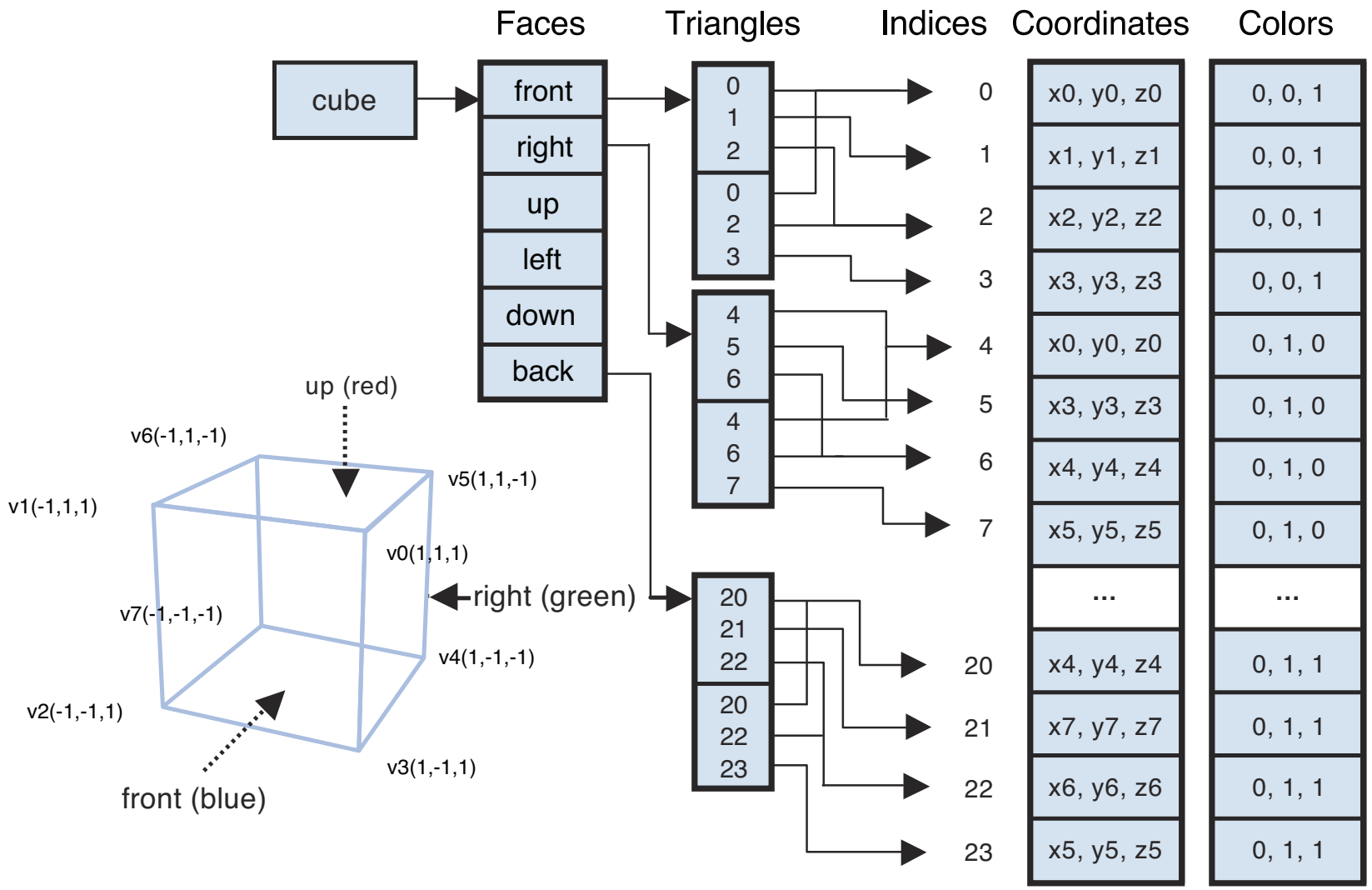


Example #10: `ColoredCube`

Example #10: `ColoredCube`

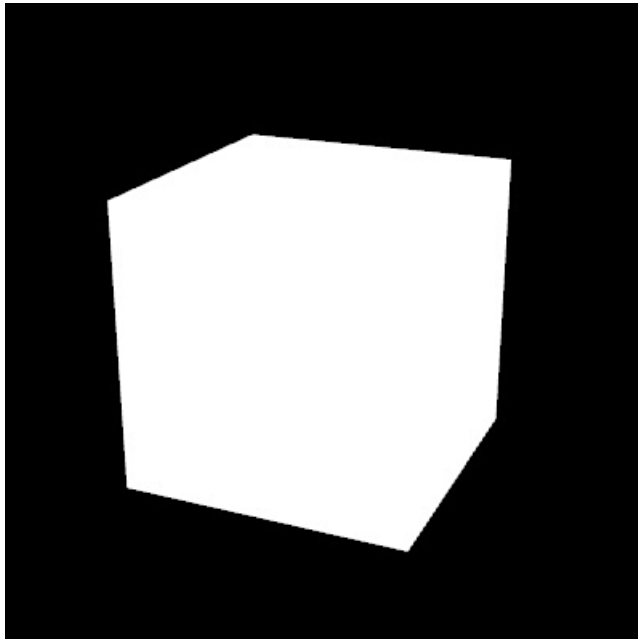
- http://rodger.global-linguist.com/webgl/ch07/ColoredCube.html
- What to learn
  - To specify different color to each face of a cube
- We need to duplicate the vertices
  → Almost no memory saving by indexed drawing
- This case is rare in real-world situations.

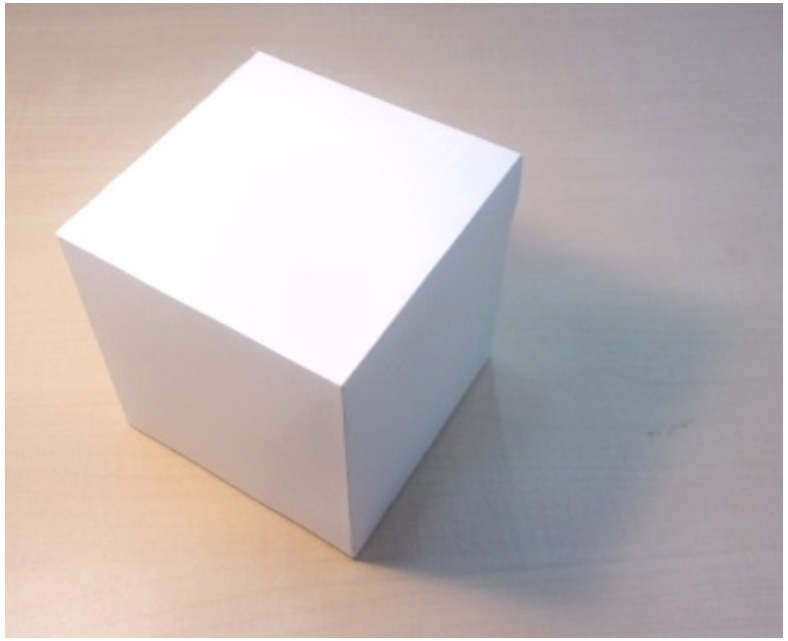
Example #10: `ColoredCube`



Lab Activities

- Try assigning a single color, e.g. white, to all the faces. What is the problem?
- http://rodger.global-linguist.com/webgl/ch07/ColoredCube_singleColor.html





Importing 3D Models

- http://rodger.global-linguist.com/webgl/ch10/OBJViewer.html
  - Wavefront OBJ format importer
  - Requires “local file access” flag when executing the browser
- 3D model galleries
  - https://poly.google.com
  - https://www.turbosquid.com
  - …and more