Introduction to Computer Graphics with WebGL

Ed Angel

Professor Emeritus of Computer Science

Founding Director, Arts, Research, Technology and Science Laboratory

University of New Mexico

Computer Viewing
Positioning the Camera

Ed Angel
Professor Emeritus of Computer Science
University of New Mexico

Objectives

- Introduce the mathematics of projection
- Introduce WebGL viewing functions in MV.js
- Look at alternate viewing APIs

From the Beginning

- In the beginning:
  - fixed function pipeline
  - Model-View and Projection Transformation
  - Predefined frames: model, object, camera, clip, ndc, window
- After deprecation
  - pipeline with programmable shaders
  - no transformations
  - clip, ndc window frames
- MV.js reintroduces original capabilities

Computer Viewing

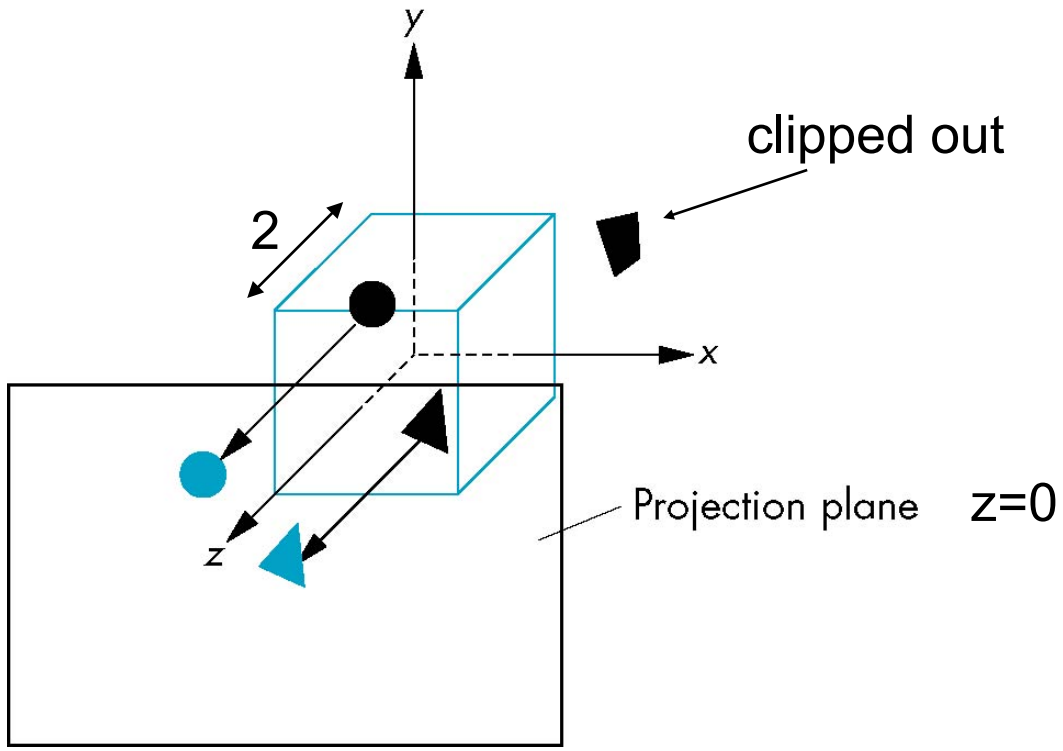
- There are three aspects of the viewing process, all of which are implemented in the pipeline,
  - Positioning the camera
    - Setting the model-view matrix
  - Selecting a lens
    - Setting the projection matrix
  - Clipping
    - Setting the view volume

The WebGL Camera

- In WebGL, initially the object and camera frames are the same
  - Default model-view matrix is an identity
- The camera is located at origin and points in the negative z direction
- WebGL also specifies a default view volume that is a cube with sides of length 2 centered at the origin
  - Default projection matrix is an identity

Default Projection

Default projection is orthogonal



Moving the Camera Frame

- If we want to visualize objects with both positive and negative z values we can either
  - Move the camera in the positive z direction
    - Translate the camera frame
  - Move the objects in the negative z direction
    - Translate the world frame
- Both of these views are equivalent and are determined by the model-view matrix
  - Want a translation (`translate(0.0,0.0,-d);`)

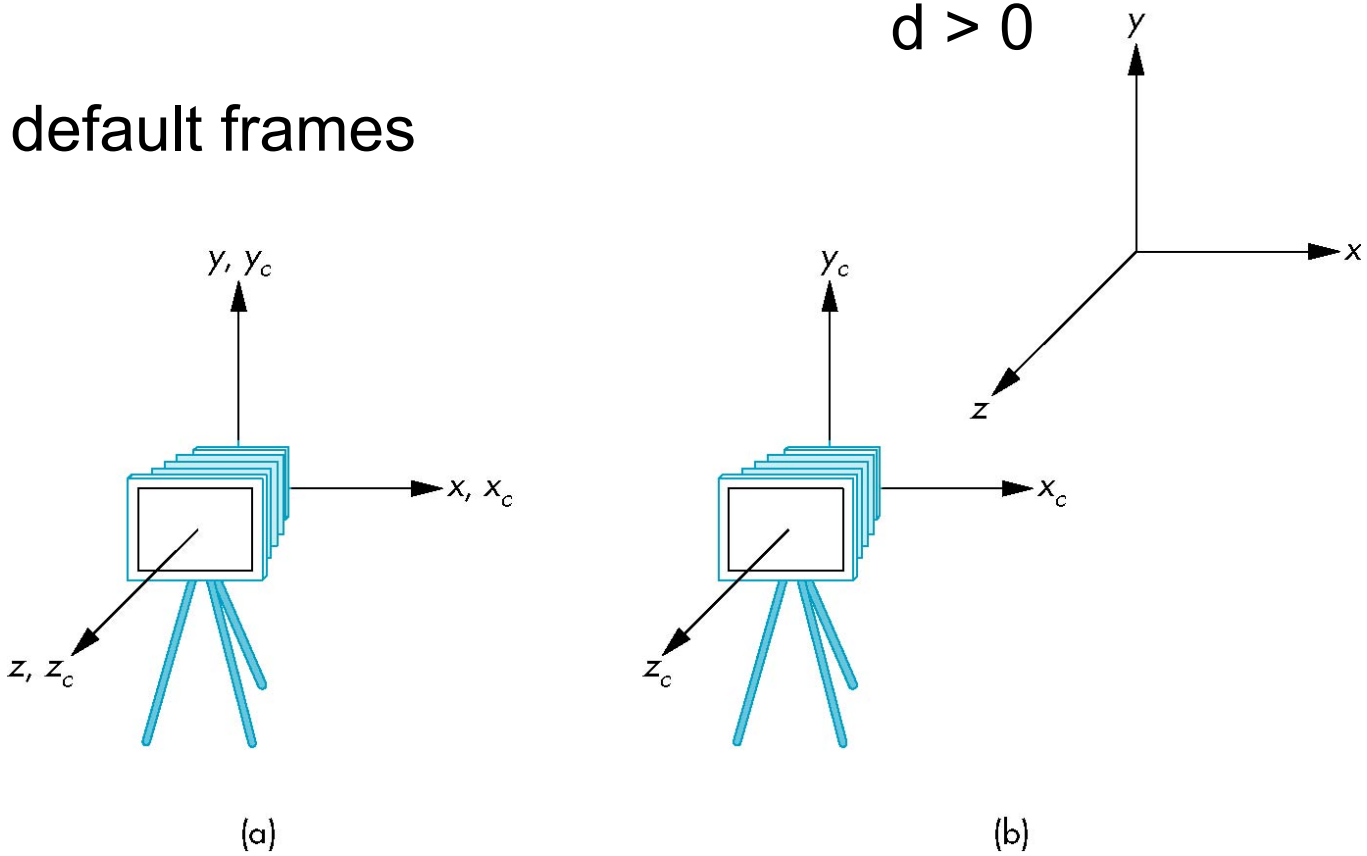
`-d > 0`

Moving Camera back from Origin

frames after translation by –d

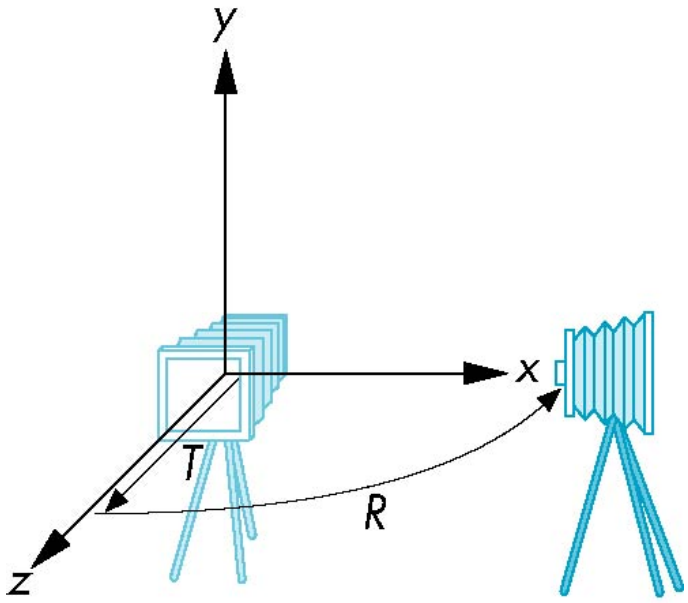
d > 0

default frames



Moving the Camera

- We can move the camera to any desired position by a sequence of rotations and translations
- Example: side view
  - Rotate the camera
  - Move it away from origin
  - Model-view matrix C = TR



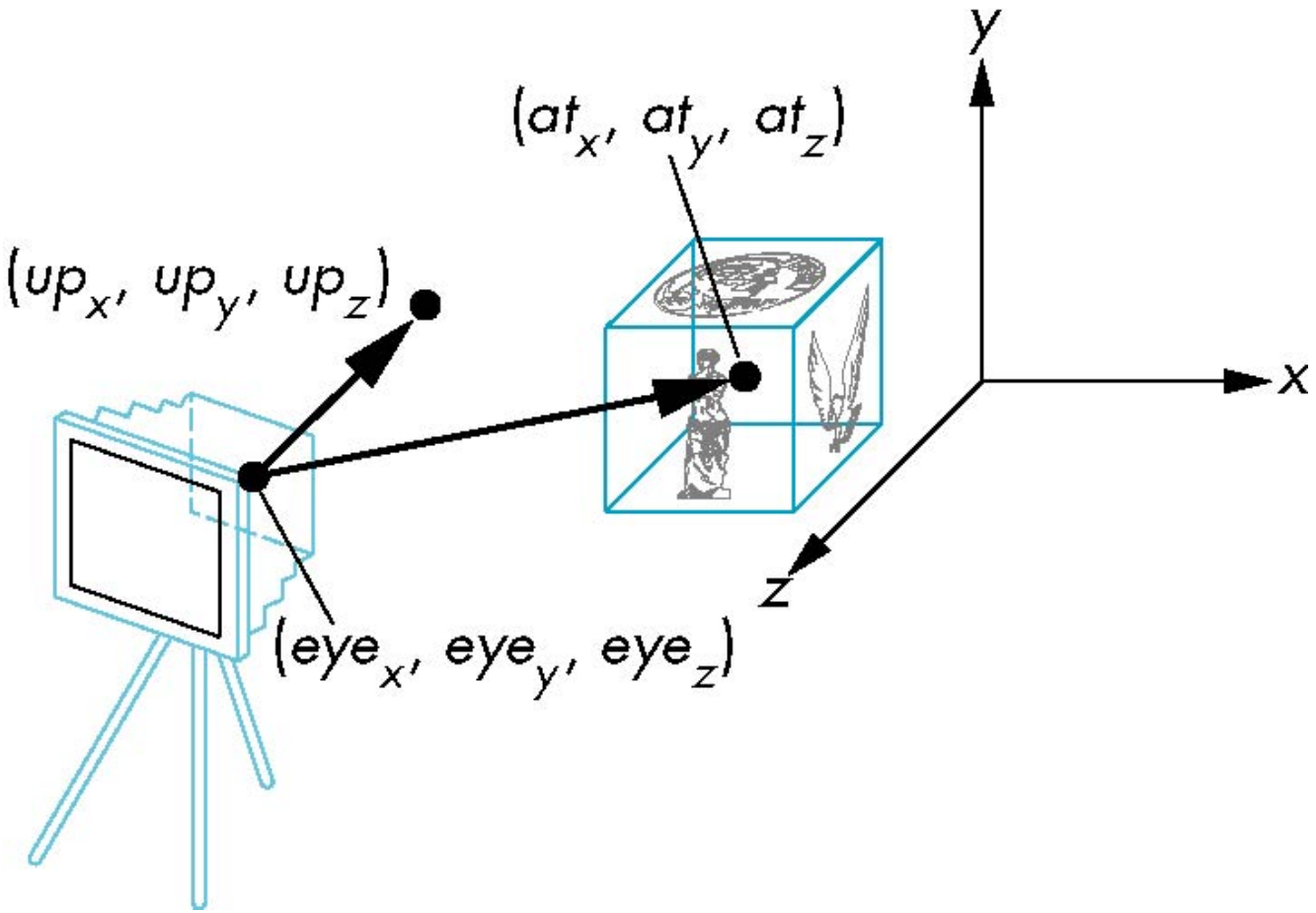
WebGL code

- Remember that last transformation specified is first to be applied

```
// Using MV.js

var t = translate (0.0, 0.0, -d);
var ry = rotateY(90.0);
var m = mult(t, ry);

or

var m = mult(translate (0.0, 0.0, -d),
         rotateY(90.0));
```

lookAt

```
LookAt(eye, at, up)
```



The lookAt Function

- The GLU library contained the function gluLookAt to form the required modelview matrix through a simple interface
- Note the need for setting an up direction
- Replaced by lookAt() in MV.js
  - Can concatenate with modeling transformations
- Example: isometric view of cube aligned with axes

```
var eye = vec3(1.0, 1.0, 1.0);
var at = vec3(0.0, 0.0, 0.0);
var up = vec3(0.0, 1.0, 0.0);

var mv = LookAt(eye, at, up);
```

Other Viewing APIs

- The LookAt function is only one possible API for positioning the camera
- Others include
  - View reference point, view plane normal, view up (PHIGS, GKS-3D)
  - Yaw, pitch, roll
  - Elevation, azimuth, twist
  - Direction angles