Computational Graphics: Lecture 26

The CVD-Lab Team

Mon, May 12, 2014

Outline: Graphics pipelines

Introduction

Camera models in `Three.js` 1/2

The rendered scene portion is contained in the "view volume"

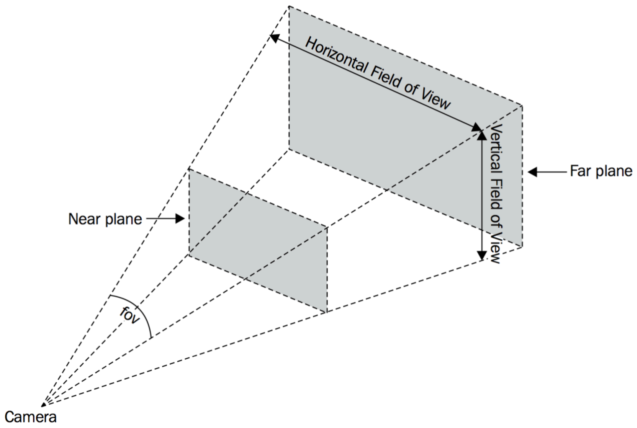


Figure : Perspective "camera" model

Camera models in `Three.js` 2/2

A scene clipping to the view volume is performed

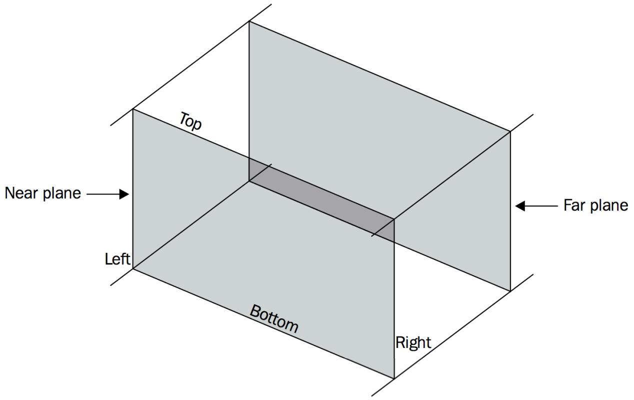


Figure : Orthographic "camera" model

2D pipeline

GKS: ISO standard 2D

Coordinate systems: World (WC), Normalized device (NDC), Device (DC)

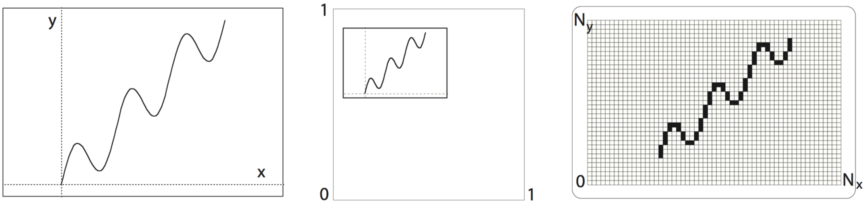


Figure 9.1 (a) Graph of the function $f : \mathbb{R} \to \mathbb{R} : x \mapsto x + 3 \sin x$, with $x \in [0, 6\pi]$, and *window* in WC (b) Normalized device coordinates and *viewport* in NDC (c) Rasterized picture in DC

Figure : A function graph from world coordinates to device coordinates

Window → Viewport transformation

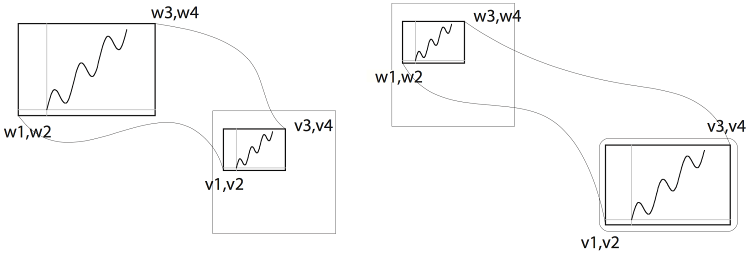


Figure 9.2 (a) Normalization transformation: WC → NDC; (b) Device transformation: NDC → DC

Device transformation Analogously, a *device transformation* $\boldsymbol{T}_D$ is a bijective affine mapping between a *workstation window* $W_{ndc} \subset NDC$ and a *workstation viewport* $V_{dc} \subset DC$:

Figure : Same mapping mechanism

Window $\rightarrow$ Non-isomorphic viewport mapping (1/3)

Direct method: computation of the unknown mapping matrix [**M**]

Direct Method We use here homogeneous coordinates. The two extreme points $(w_1, w_2, 1)^T$ and $(w_3, w_4, 1)^T$ of window W are respectively mapped to the two extreme points $(v_1, v_2, 1)^T$ and $(v_3, v_4, 1)^T$ of viewport V. Also, the $\boldsymbol{M}$ tensor must transform a 2D extent, parallel to the reference frame, onto another one of the same kind. Hence, the matrix $[\boldsymbol{M}]$ will have a predictable structure, with only (unknown) coefficients of scaling and translation:

$$\begin{pmatrix} v_1 & v_3 \\ v_2 & v_4 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} a & 0 & b \\ 0 & c & d \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} w_1 & w_3 \\ w_2 & w_4 \\ 1 & 1 \end{pmatrix}$$

Figure : Only four unknown coefficients in [**M**]

Window $\rightarrow$ Non-isomorphic viewport mapping (2/3)

Direct method: derivation of the unknown coeffs in [**M**]

The previous matrix equation is equivalent to four scalar simultaneous equations in the unknown coefficients a, b, c and d:

$$\left\{ \begin{array}{rcl} a\, w_1 + b & = & v_1 \\ c\, w_2 + d & = & v_2 \\ a\, w_3 + b & = & v_3 \\ c\, w_4 + d & = & v_4 \end{array} \right. , \quad \text{and hence} \quad \left\{ \begin{array}{rcl} a & = & \frac{v_3 - v_1}{w_3 - w_1} \\ c & = & \frac{v_4 - v_2}{w_4 - w_2} \\ b & = & v_1 - \frac{v_3 - v_1}{w_3 - w_1} w_1 \\ d & = & v_2 - \frac{v_4 - v_2}{w_4 - w_2} w_2 \end{array} \right.$$

Figure : Finally substitute the expressions for a, b, c, d in [**M**]

Window $\rightarrow$ Non-isomorphic viewport mapping (3/3)

A different approach to compute $\mathbf{M} : W \rightarrow V$

Composition of elementary transformations The window-viewport mapping tensor $\boldsymbol{M}$ can be derived easily by composition of some elementary transformations:

1. a translation $\boldsymbol{T}_1$ which maps the point (w_1, w_2) into the origin $\boldsymbol{o}$ of the reference system;
2. a scaling $\boldsymbol{S}_1$ of W onto the standard unit square;
3. a scaling $\boldsymbol{S}_2$ of the standard unit square onto V;
4. a translation $\boldsymbol{T}_2$ which maps $\boldsymbol{o}$ into (v_1, v_2).

In other words we have:

$$\boldsymbol{M} : W \rightarrow V : \quad \boldsymbol{p} \mapsto (\boldsymbol{T}_2 \circ \boldsymbol{S}_2 \circ \boldsymbol{S}_1 \circ \boldsymbol{T}_1)(\boldsymbol{p}),$$

where

$$\begin{aligned} \boldsymbol{T}_1 &= \boldsymbol{T}(-w_1, -w_2), \\ \boldsymbol{S}_1 &= \boldsymbol{S}\left(\frac{1}{w_3 - w_1}, \frac{1}{w_4 - w_2}\right), \\ \boldsymbol{S}_2 &= \boldsymbol{S}\left(v_3 - v_1, v_4 - v_2\right), \\ \boldsymbol{T}_2 &= \boldsymbol{T}(v_1, v_2). \end{aligned}$$

Isomorphic viewport mapping

When the aspect ratio of window and viewport are different

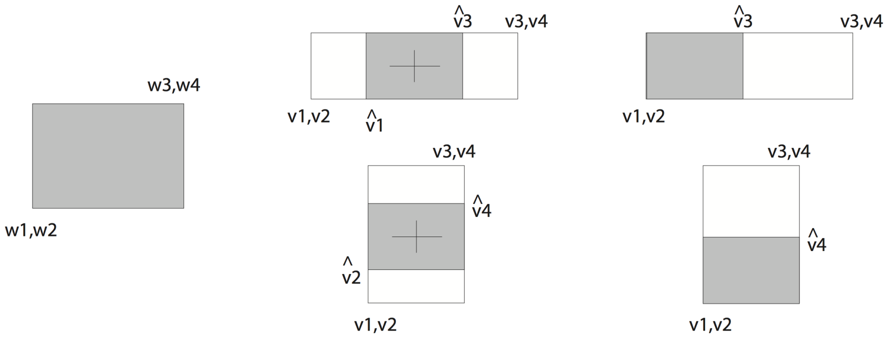


Figure 9.3 Computed viewports for preserving aspect ratio in window-viewport mapping. Two diverse strategies

Figure : Substitute a computed viewport (gray) to the user viewport (white)

Correction of window-viewport device transformation

When the second axis of DC is pointing bottom

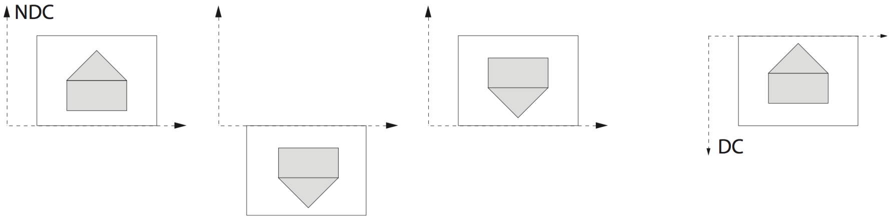


Figure 9.4 (a) Device transformation, with a reversed orientation of y axis
(b) Display on a reversed y-axis device

Figure : To reverse the axis: apply a mirroring followed by a translation

3D pipeline

The rise of OpenGL and the decline of PHIGS

From Wikipedia

OpenGL, unlike PHIGS, was an immediate-mode rendering system with no "state"; once an object is sent to a view to be rendered it essentially disappears. Changes to the model had to be re-sent into the system and re-rendered, a dramatically different programming mindset. For simple projects, PHIGS was considerably easier to use and work with.

However, OpenGL's "low-level" API allowed the programmer to make dramatic improvements in rendering performance by first examining the data on the CPU-side before trying to send it over the bus to the graphics engine. For instance, the programmer could "cull" the objects by examining which objects were actually visible in the scene, and sending only those objects that would actually end up on the screen. This was kept private in PHIGS, making it much more difficult to tune performance, but enabling tuning to happen "for free" within the PHIGS implementation.

Given the low performance systems of the era and the need for high-performance rendering, OpenGL was generally considered to be much more "powerful" for 3D programming.

3D View Model

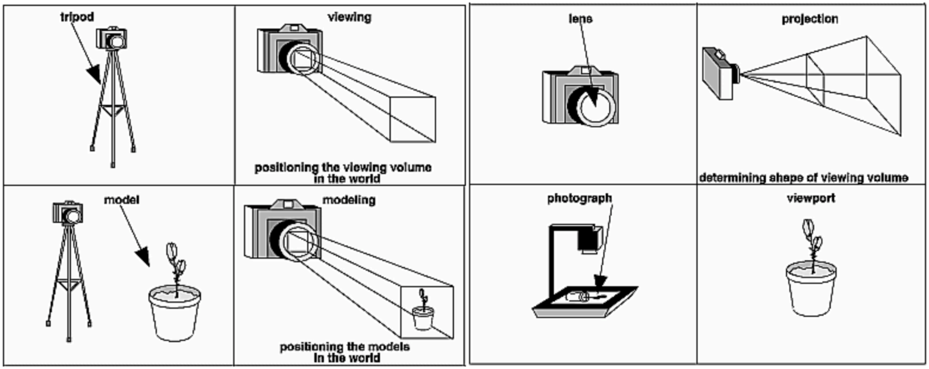
we make reference to ISO standard PHIGS. Similar concepts in OpenGL

View parameters Four 3D vectors, defined in WC, are called *view parameters* in ANSI Core. They completely specify the picture resulting from a specific projection, also called *view*, of the scene. The picture resulting from a projection is returned in a reference system linked to the projection plane. This system is called *uvn* or *view system* in ANSI Core, and *view reference* in PHIGS. A *view model* is a set of values for view parameters. The four vector parameters which specify the projection, i.e. the *view*, are the following:

Figure : 4 vector parameters and 6 scalar parameters give the view model

Overview: The Camera Analogy in OpenGL (fixed pipeline) (1/2)

From OpenGL "red book"


tripod
viewing
positioning the viewing volume in the world
lens
projection
determining shape of viewing volume
model
modeling
positioning the models in the world
photograph
viewport

Overview: The Camera Analogy in OpenGL (fixed pipeline) (2/2)

From OpenGL ”red book”

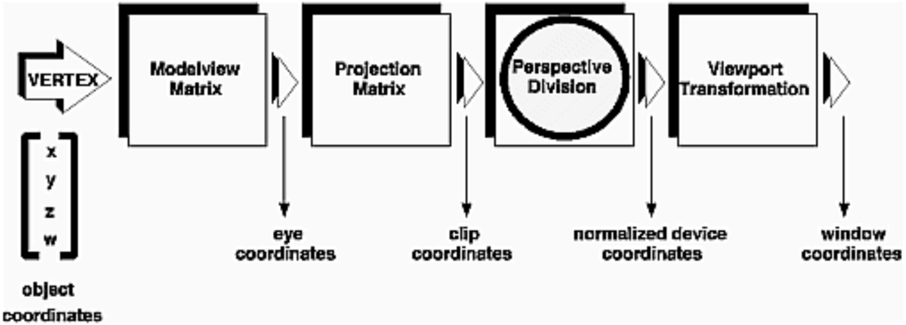


Figure : Five coordinate systems, and four coordinate transformations

The Camera Analogy in PHIGS

1. *Center of Projection* (COP) is the common point of projecting lines. It coincides with the observer's position. It is substituted by the vector called *Direction of Projection* (DOP) for parallel projections, where the projection center is improper, i.e. is set at infinity.
2. *View Reference Point* (VRP) is the point targeted by the observer, at the intersection of the view axis and the view plane. It is assumed as the origin of the view reference system.
3. *View Up Vector* (VUV) is a vector used to orientate the projected picture. The v axis of view system is parallel to the projection of view up vector.
4. *View Plane Normal* (VPN) is a vector normal to the view plane. It is assumed as the direction of the n axis of view reference system.

Figure : Four vector parameters in WC (warning on the first)

The PHIGS pipeline

A pipeline of four transformations of coordinates is associated with the five reference systems of PHIGS systems:

1. Structure Network Traversal
2. View Orientation
3. View Mapping
4. Workstation Transformation.

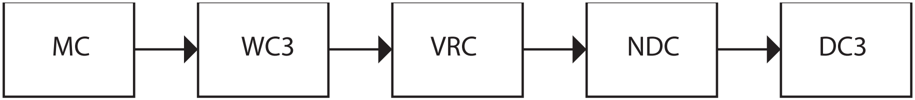
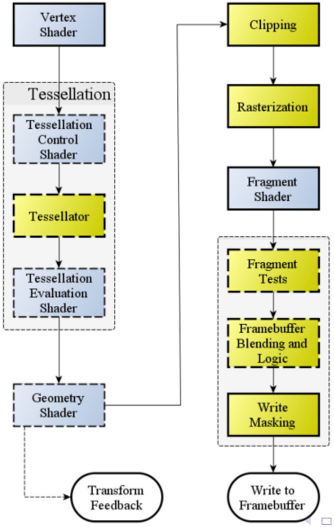


Figure : Compare with OpenGL: what differences?

OpenGL: the new programmable pipeline

OpenGL Rendering Pipeline Flowchart



References

GP4CAD book