Computer graphics 2: Graduate seminar in computational aesthetics

Angus Forbes

evl.uic.edu/creativecoding/cs526

Homework 2 – RJ ongoing...

- Follow the suggestions in the Research Journal handout and find and describe three papers OR projects of your choice. (include images, videos, diagrams)
- Come up with (or refine) five interesting ideas in computer graphics or new media arts (include text and sketches)
- Research one academic or creative venue

Homework 2 – Emulation Project

- Choose one artist from the Inventing Abstraction website

a) recreate or extend the artwork with as much detail as possible (using Three.js), focusing on shape, color, texture (due 9/15)

b) create a program that *applies* the style of the artist to any input image (or video), focusing on the use of GPU shaders (due 9/22)

Today's class

0. Discussion of papers from RJ
1. Small group discussion – ideas
2. Rendering pipeline
3. Lab session – shader programming

Rendering pipeline

- Fundamental functionality of modern computer graphics libraries is to transform objects in a user-defined 3D space onto a 2D screen.

- Using OpenGL and GLSL, you have complete control over this transformation and how to define (“shade”) each pixel of your 3D geometry.

CPU / GPU

CPU side

Define geometry; handle interaction

GPU side

Vertex shader: transform geometry from 3D space to 2D space

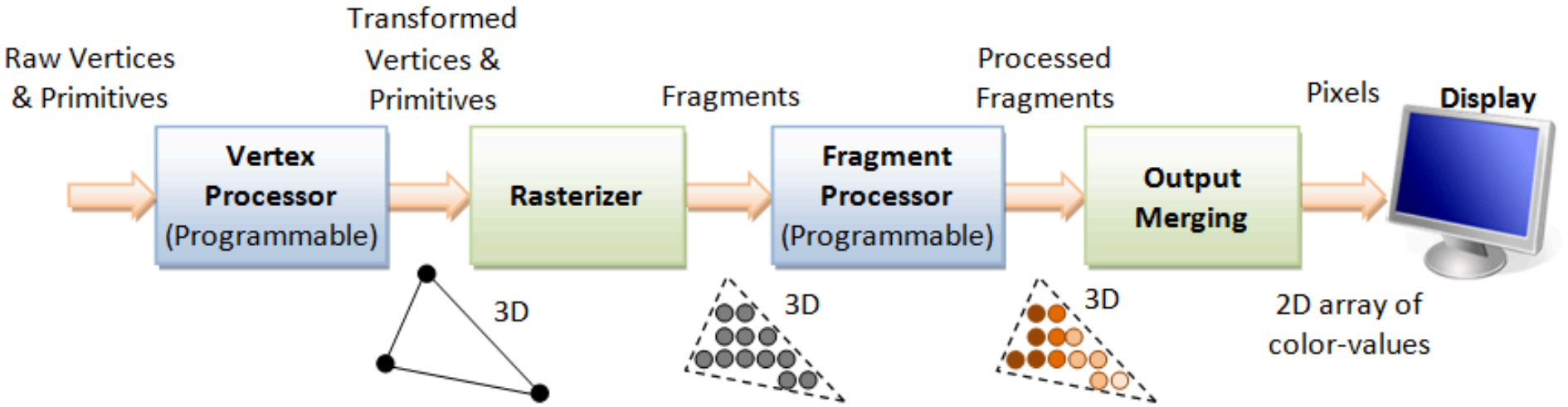
Fragment shader (also called a Pixel shader): determine pixel color of every pixel inside the geometry

Geometry

All 3D objects are defined as one of three geometric primitives: points, lines, or triangles (and usually triangles).

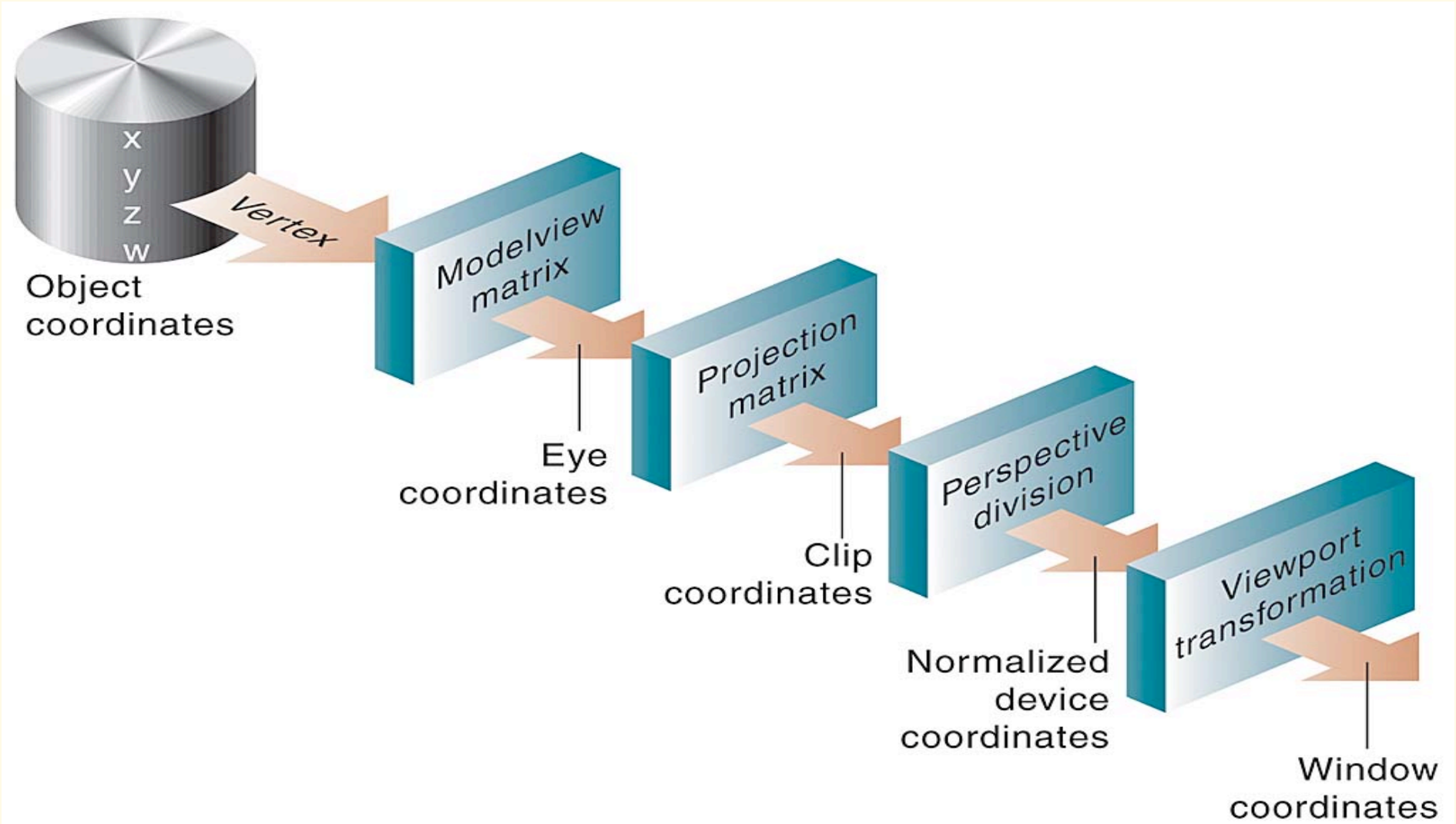
This geometry, or vertex data, is passed into the GPU via special OpenGL/WebGL commands that bind a specified shader program on the GPU and matching terms on a vertex shader.

Geometry



3D Graphics Rendering Pipeline: Output of one stage is fed as input of the next stage. A vertex has attributes such as (x, y, z) position, color (RGB or RGBA), vertex-normal (n_x, n_y, n_z), and texture. A primitive is made up of one or more vertices. The rasterizer raster-scans each primitive to produce a set of grid-aligned fragments, by interpolating the vertices.

Rendering pipeline



Coordinate systems

The *Object* or *Local* coordinate system is defined in terms of the Geometry itself. The origin is usually the center or the lower-left of the object.

Coordinate systems

The *Model* or *World* coordinate system defines the x, y, and z axes which serve as a basis for the 3D space. Where is the origin? Which way is up?

Coordinate systems

The *Eye*, *Camera*, or *View* coordinate system defines another set of x, y, and z axes which server as a different basis for the 3D space. The camera is always positioned at the origin of this coordinate system.

Coordinate systems

The *Clip* coordinate system describes the bounded view of the visible by the camera in terms of both the "lens" of the camera, its "depth of focus", and the aspect ratio of the screen bounds.

Coordinate systems

The *Normalized Device* coordinates is the same view normalized from -1 to +1 along each axis.

Coordinate systems

The *Window* or *Screen* coordinates are these x and y coordinates positioned within the screen bounds. The z is used for depth-testing and is bound between 0 and 1.

Setting up coordinate systems

4x4 matrices are used to encode transformations between each of these coordinate systems.

3d – 2d normalized

- Model transformation
- View transformation
- Projection transformation

2d normalized - pixel

- Viewport definition

Setting up coordinate systems

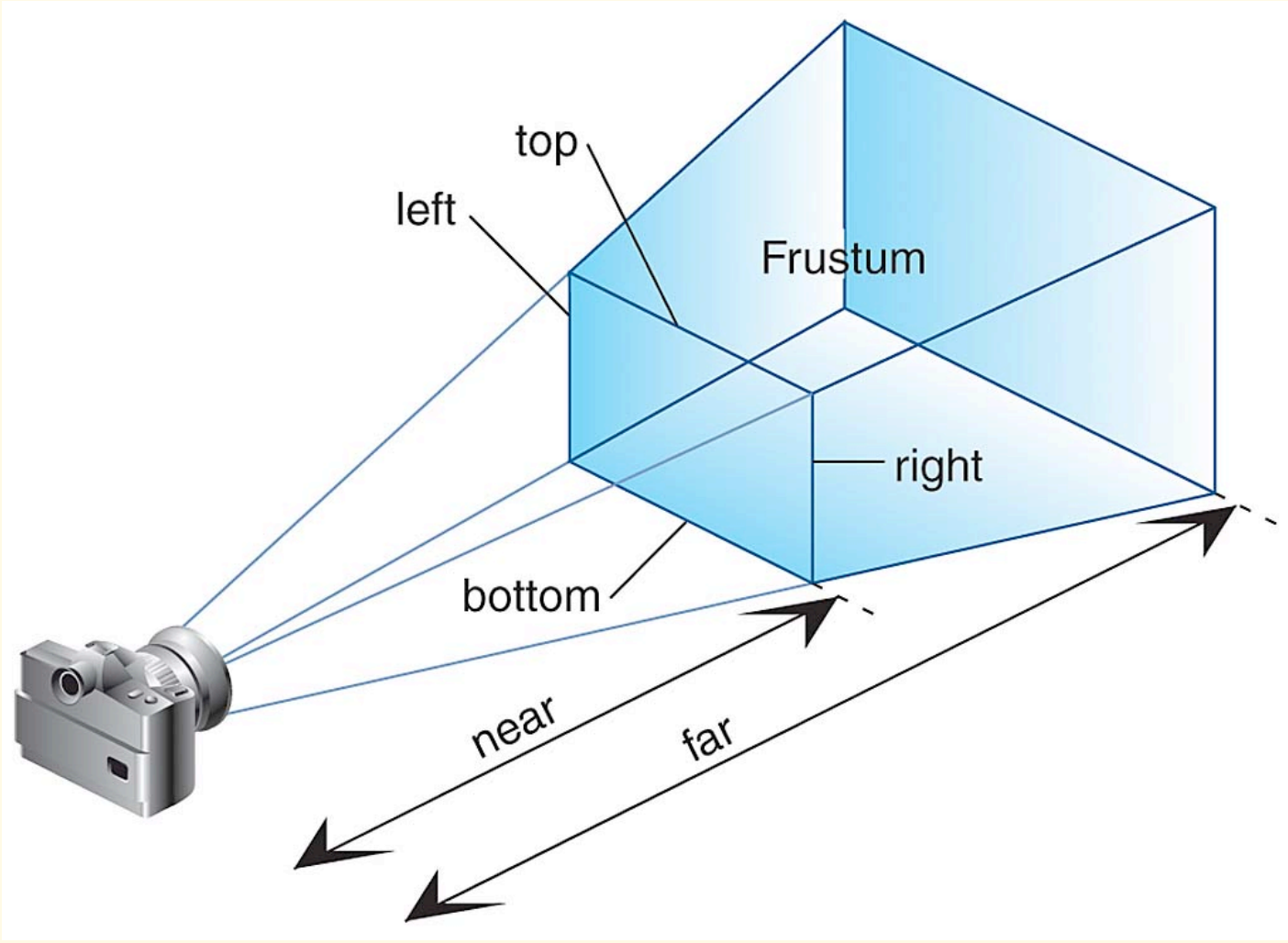
A vertex is left multiplied through each matrix to transform it into the new coordinate system.

Modelview

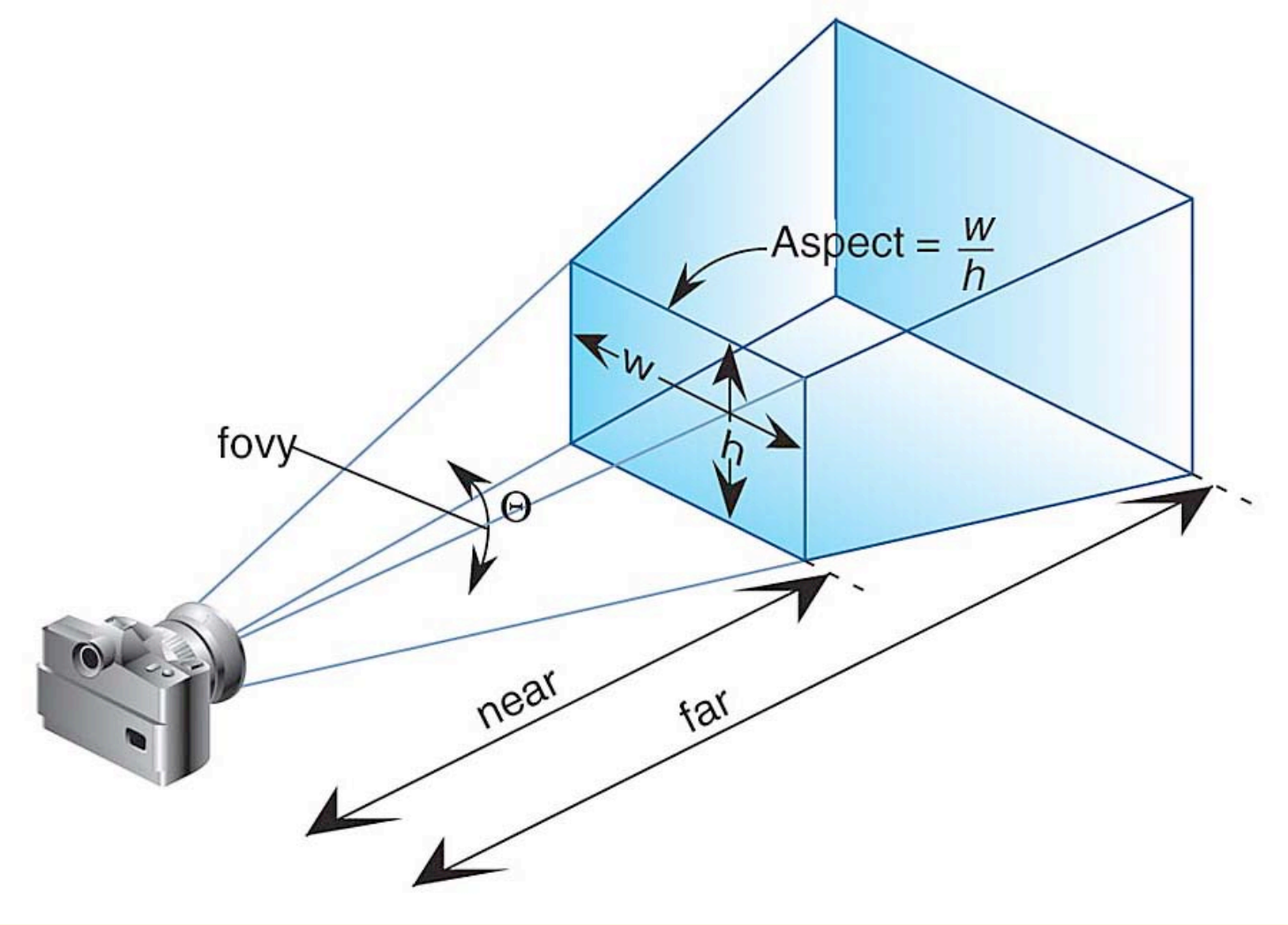
Sometimes the model and view transformations are encoded in a single matrix, MV = V * M.

Sometimes the model, view, and projection transformations are encoded in a single matrix, MVP = P * V * M.

Perspective projection



Perspective projection



Lab session, Three.js

Create a scene containing a series of geometric objects, use a GLSL shader program to displace the vertices and changes the pixel colors.

Three.js, GLSL built-ins

geometry.vertices[i].position =>

    position

object.matrixWorld =>

    modelMatrix

camera.projectionMatrix =>

    projectionMatrix

camera.matrixWorldInverse =>

    viewMatrix

camera.matrixWorldInverse * object.matrixWorld =>

    modelViewMatrix

https://github.com/mrdoob/three.js/issues/1188

Light Space Modulator, Moholy-Nagy

http://
www.sfmoma.org/
explore/
multimedia/
videos/1