Interactive Computer Graphics

Dr Bernhard Kainz

b.kainz@imperial.ac.uk

Huxley 372

Intro

- https://www.youtube.com/watch?v=wAu8w7n4LHM

Interactive Computer Graphics

- Please note that this course has been timetabled for 2 hours per week:
  - Tuesday 9-10, Zoom
  - Wednesday 9-10, MS Teams

- However, not all timetabled slots will be used every week so **please check the timetable** on the webpage for more information:

  http://wp.doc.ic.ac.uk/bkainz/teaching/60005-co317-computer-graphics/

Interactive Computer Graphics

- Printouts:
  - Lecture notes & tutorials:
    - Please print your own if you want a hardcopy

- Lectures:
  - All lectures have slides that are available via CATE
  - Some lectures (not all) have notes that are available via CATE
  - Lectures are pre-recorded, we will have a weekly Q&A session

- Tutorials:
  - All tutorials have sample solutions that are available a few days after the tutorial.

Interactive Computer Graphics

- ## Course overview:
  - Syllabus, timetable and news on

    http://wp.doc.ic.ac.uk/bkainz/teaching/60005-co317-computer-graphics/

  - See notes on vector algebra revision (link)

- ## Course materials and notes:
  - Look at CATE for lecture notes, tutorials & coursework

Information for non DOC students

- Apply at https://dbc.doc.ic.ac.uk/externalreg/

- Your department's endorser will approve/reject your application
  - No access after a few days? Check status of approval and contact relevant person(s)

- Key Dates:
  - Exam registration opens end January for 2-3 weeks
  - Exams for DoC 3rd/4th year courses take place at the end of the Term in which the course is taught – courses that are co-scheduled on the time-table will have their exams co-scheduled

- If in doubt, read the guidelines available at the link above

Courseworks

- There will be six practical coursework tasks; three of them are assessed:
  - Task 1: Framework
  - Task 2: Transformations
  - Task 3: Illumination (assessed 40%)
  - Task 4: Color
  - Task 5: Texture & Render to Texture (assessed 10%)
  - Task 6: GPU ray tracing (50%)

- All practical courseworks require programming experience (very basic C)

Logistics

- 6 tasks, 3 assessed
  - 1-5 One per week
  - Task 6: 2 weeks
- Description and framework already available for all exercises, but
- Necessary knowledge in each lecture per week
- Submission electronically via CATE!

Effects previous year's SOLE and COVID-19

- Redesigned the coursework to better match the content of the lectures in each week
- Made the framework available to everybody through a browser implementation (no computing lab requirement anymore)
- Provide an open-source implementation of a custom OpenGL GLSL IDE
- Reduced the workload to three assessed tasks, revising assessment. Tasks 1,2,4 are voluntary. Removed one task that was not supported by all OpenGL versions
- One exam question will be based on what you learned during the coursework!
- Re-implemented the framework for the coursework for a second time:
  - It is now the most advanced teaching framework for computer graphics
- Everybody can use it now from their own laptops without needing to install anything: http://shaderlabweb.doc.ic.ac.uk/
- We listened to your SOLE feedback from the last years!
  - Please fill in SOLE at the end of this course!

CSL and TAs

- Course support leader:
    - Benjamin Hou benjamin.hou11@imperial.ac.uk
    - Samuel Budd
    - Hadrien Reynaud
    - Miguel Monteiro

Labs in 202 & 206

- Week 3: Wednesday 9-10
- Week 4: Wednesday 9-10
- Week 6: Wednesday 9-10
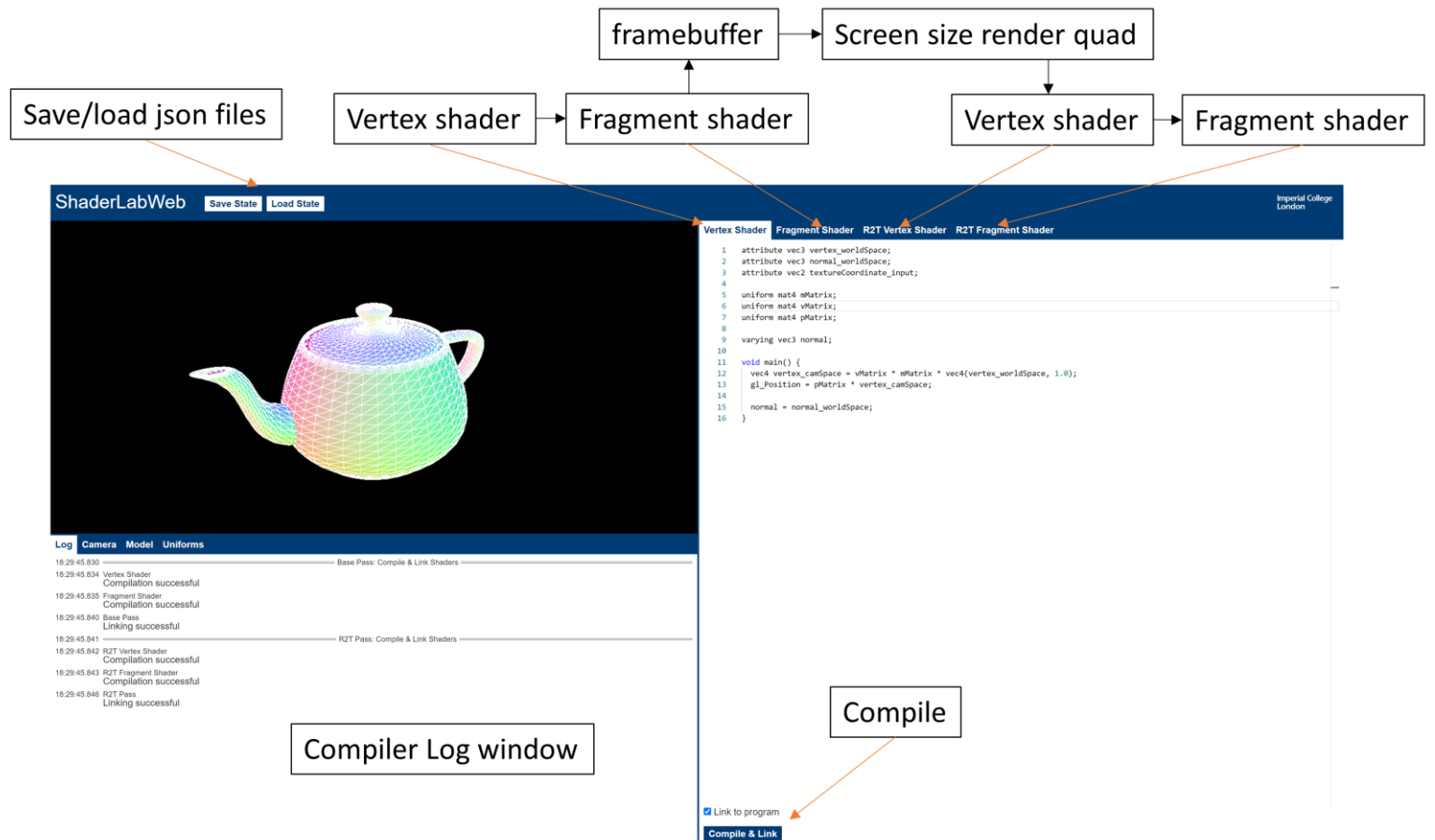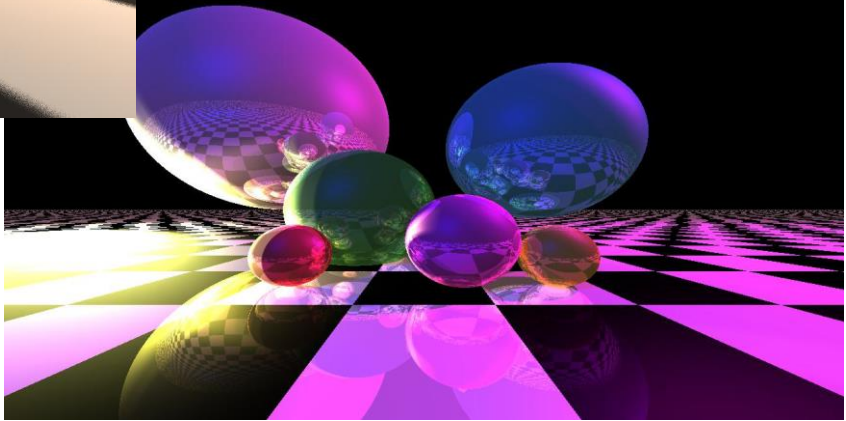- Week 8: Wednesday 9-10
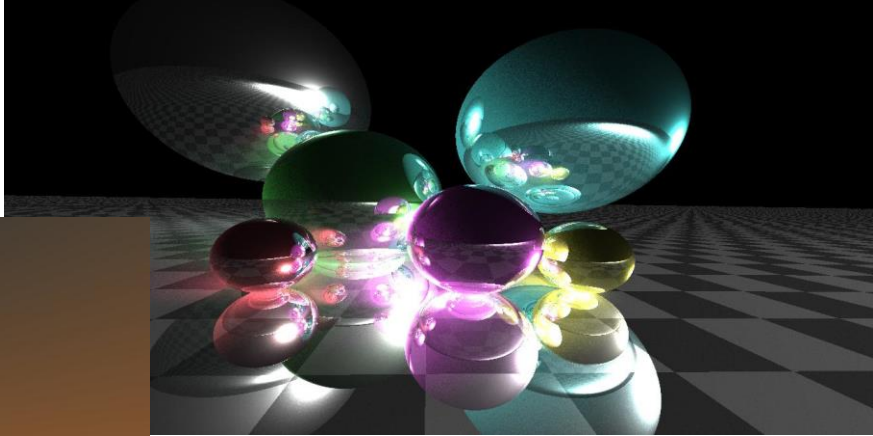
framework

- http://shaderlabweb.doc.ic.ac.uk/
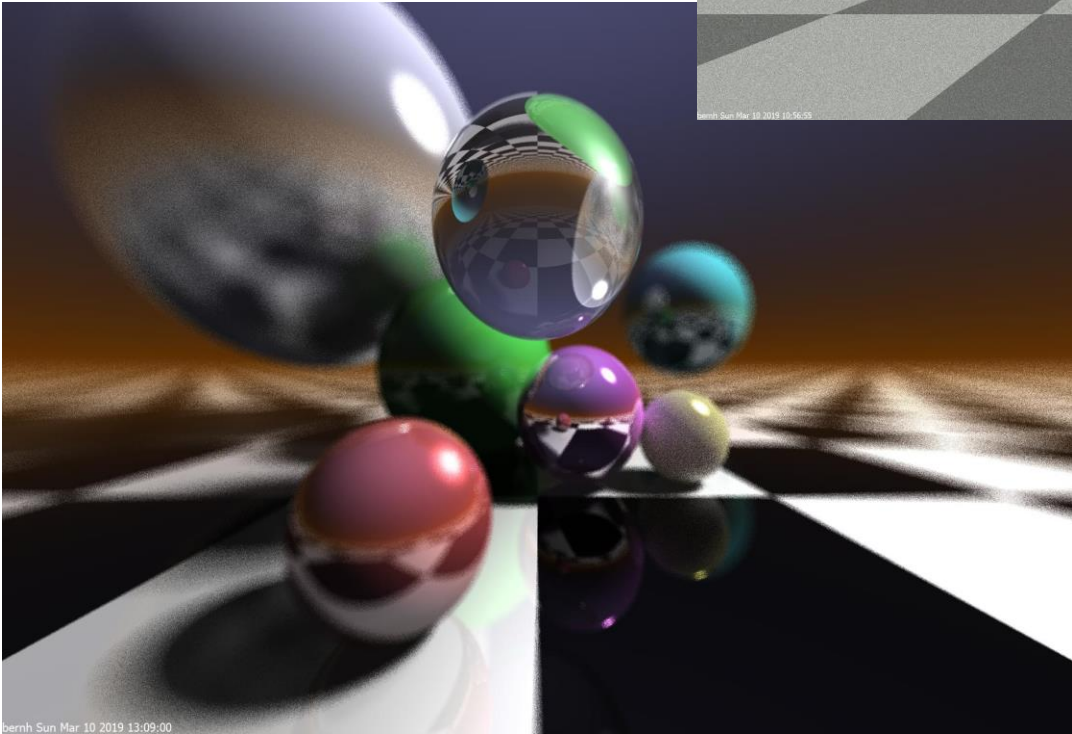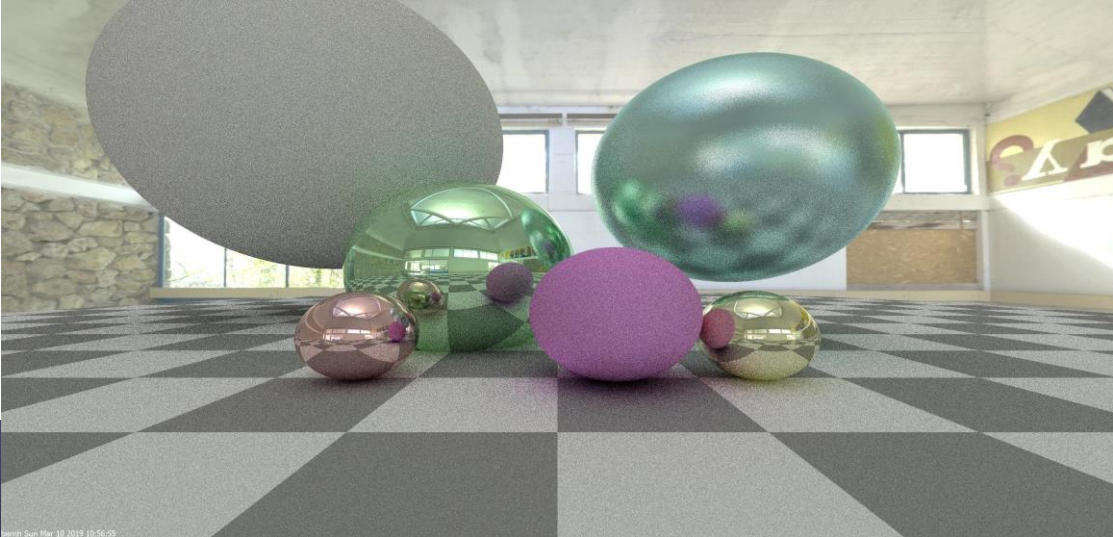
starting the framwork

- Open a browser (preferably Chrome)
- Enter http://shaderlabweb.doc.ic.ac.uk/

Tasks

- Task 1:
  Get familiar with the framework

Student solutions
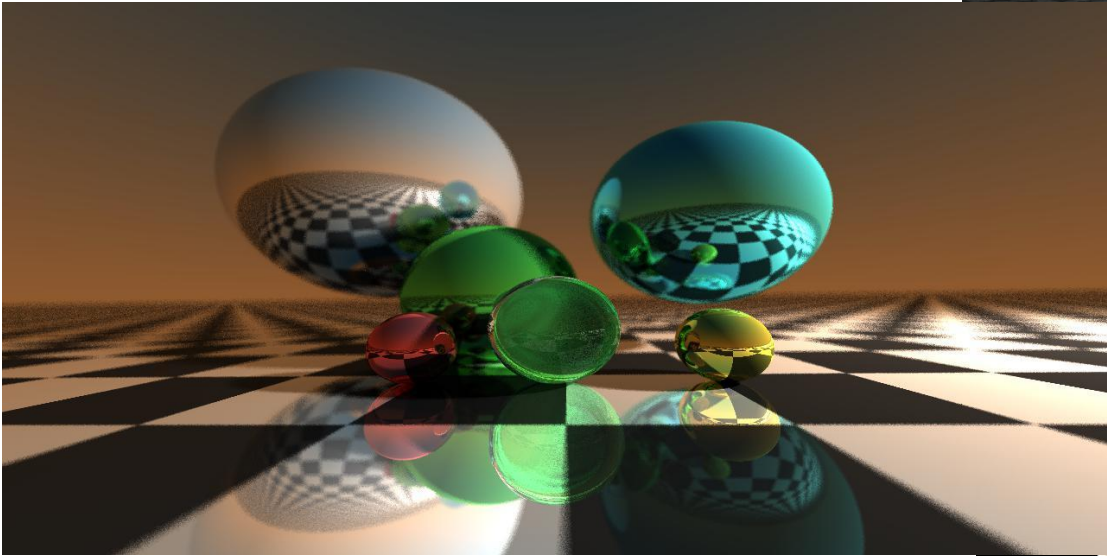
Student solutions

Student solutions

Questions:
https://edstem.org/us/courses/14755/discussion/
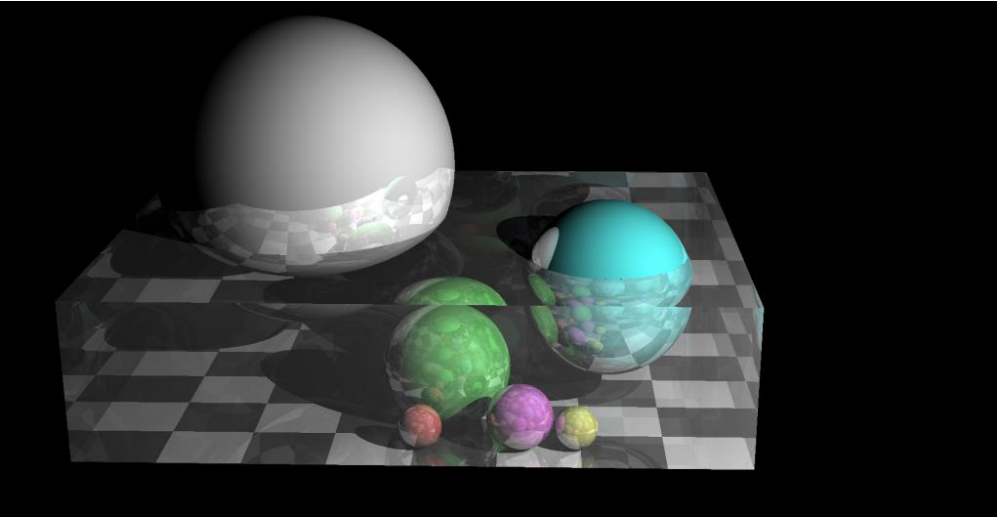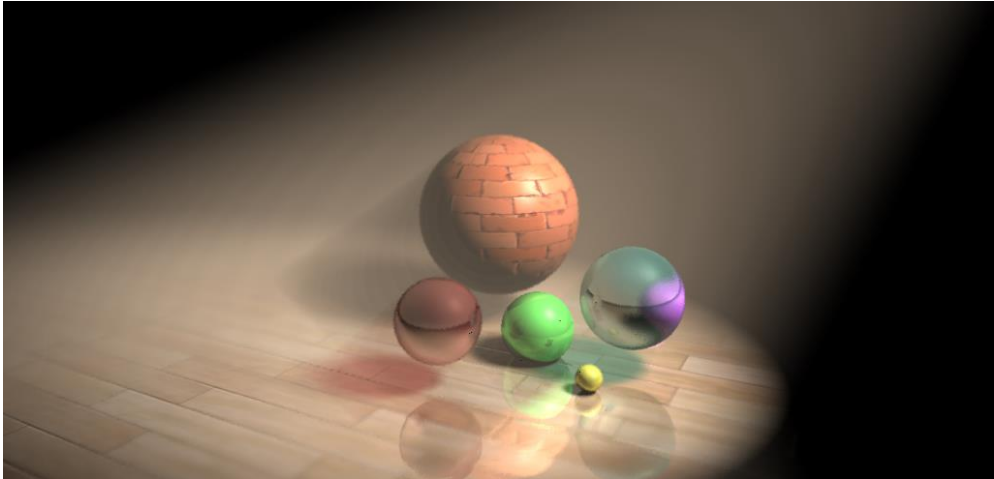
Have fun!

Interactive Computer Graphics: Lecture 1

3D graphical scenes:

Projections and Transformations

Two dimensional graphics

- The lowest level of graphics processing operates directly on the pixels in a window provided by the operating system.

- Typical Primitives are:

```
SetPixel(int x, int y, int colour);
DrawLine(int xs, int ys, int xf, int yf);
```

- etc.

World coordinate systems

- To achieve *device independence* when drawing objects we can define a **world coordinate system.**

- This will define our drawing area in units that are suited to the application:

  - meters
  - light years
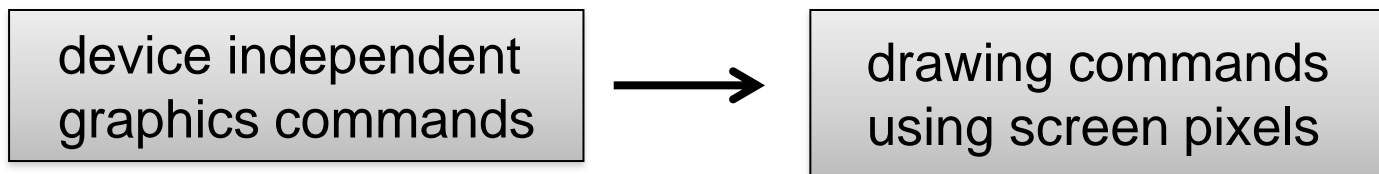 - microns
  - etc

Example

We can give our window 'World Coordinates' and draw objects using them.

```
SetWindow(30, 10, 70, 50)
DrawLine(40, 3, 90, 30)
DrawLine(50, 60, 60, 40)
```
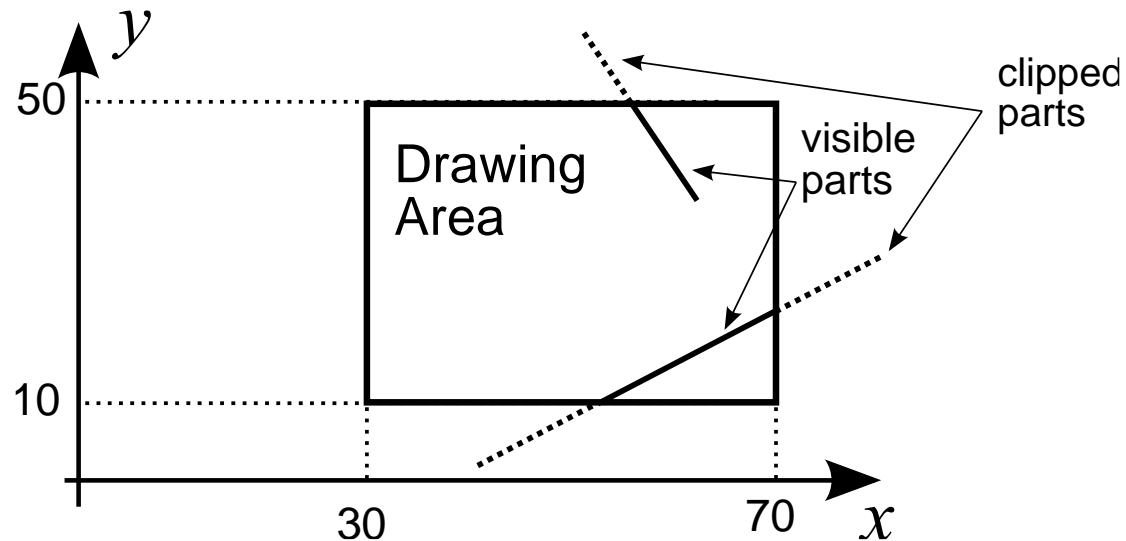
Normalisation

To make the conversion

| device independent graphics commands	→	drawing commands using screen pixels

we need a process of normalisation

First we must ask the operating system* for the pixel addresses of the corners of the area we are using.

Then we can translate our world coordinates to pixel coordinates.

*making a 'system call' through the API

Normalisation

World coordinate
window

(X_w, Y_w)

Screen

(X_v, Y_v)

Viewport
(Pixel coordinates)

W_{xmin} W_{xmax} V_{xmin} V_{xmax}
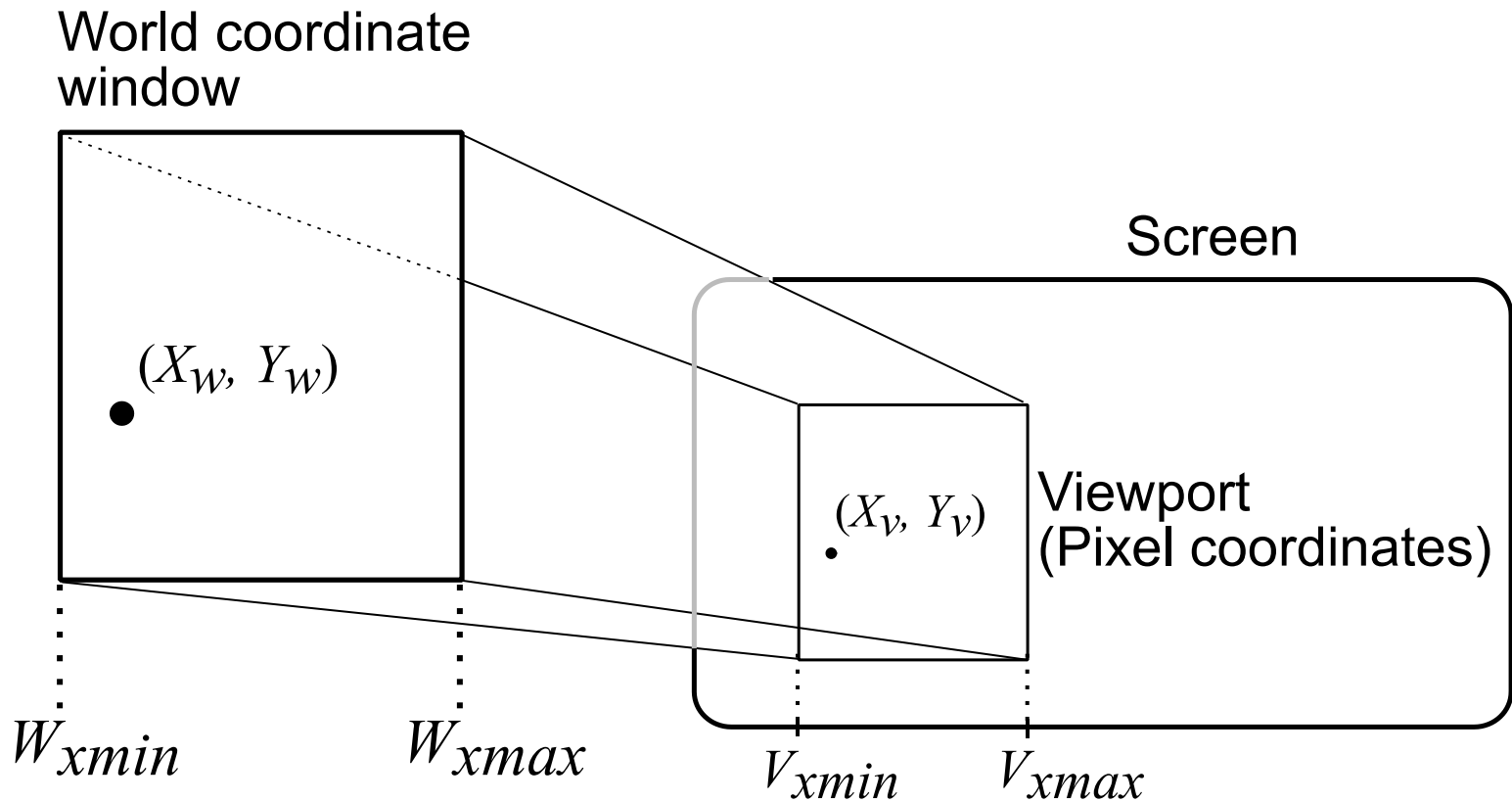
Normalisation

- Having defined our world coordinates, and obtained our device coordinates we relate the two by simple ratios:

$$\frac{(X_w - W_{xmin})}{(W_{xmax} - W_{xmin})} = \frac{(X_v - V_{xmin})}{(V_{xmax} - V_{xmin})}$$

- Rearranging, we get:

$$X_v = \frac{(X_w - W_{xmin})(V_{xmax} - V_{xmin})}{W_{xmax} - W_{xmin}} + V_{xmin}$$

- with a similar expression for Y_v

World coordinate window

Screen

(X_w, Y_w)

(X_v, Y_v) Viewport (Pixel coordinates)

W_{xmin} W_{xmax} V_{xmin} V_{xmax}
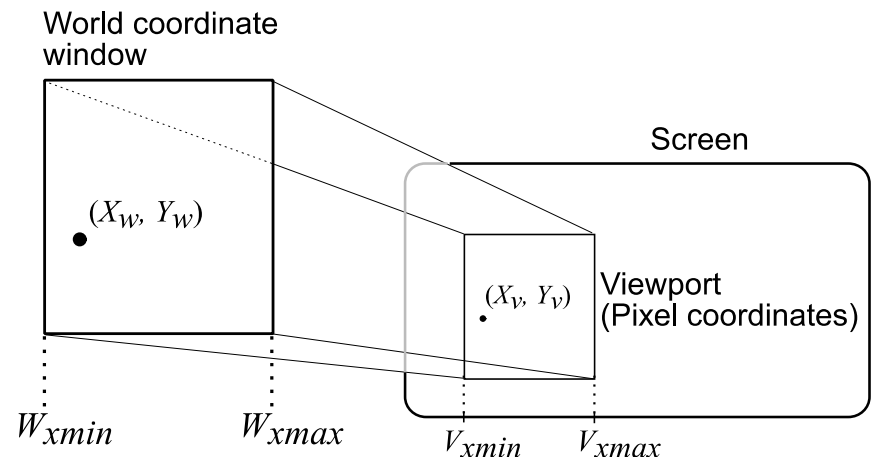
Normalisation

- So we have two equations for calculating pixel coordinates (X_v, Y_v).

- We can simplify them to form a simple pair of linear equations:

$$
\begin{aligned}
X_v &= A X_w + B \\
Y_v &= C Y_w + D
\end{aligned}
$$

- Here A, B, C and D are constants that define the normalisation. A, B, C, D are found from the known values of W_{xmin}, V_{xmin}, ...

Polygon rendering

- Many graphics applications use scenes built out of planar polyhedra.

- These are three dimensional objects whose faces are all planar polygons (often called <u>faces</u> or <u>facets</u>).
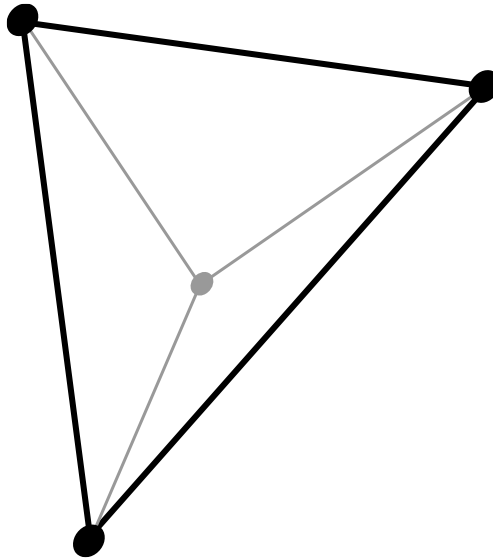
Representing planar polygons

- In order to represent planar polygons in the computer we need a mixture of different data:
  - Numerical Data
    - Actual 3D coordinates of vertices, etc.
  - Topological Data
    - Details of what is connected to what.
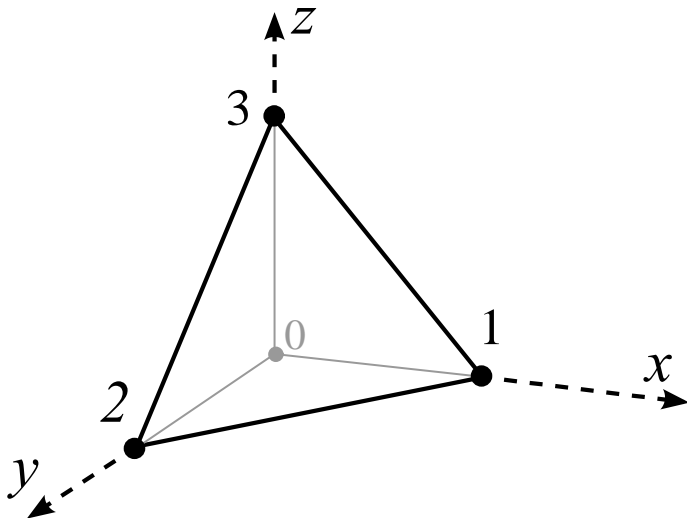
Vertex data	
Index	Location
0	(0, 0, 0)
1	(1, 0, 0)
2	(0, 1, 0)
3	(0, 0, 1)

Face data	
Index	Vertices
0	0 1 3
1	0 2 1
2	0 3 2
3	1 2 3

Projections of wire frame models

- Wire frame models simply include points and lines.
- In order to draw a 3D wire frame model we must:
    - First convert the points to a 2D representation.
    - Then we can use simple drawing primitives to draw them.
- The conversion from 3D into 2D is a *projection*.
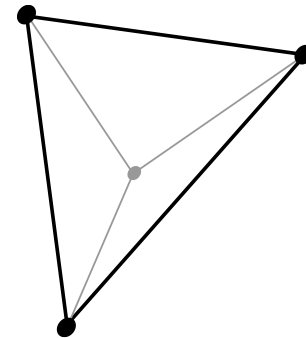
Projection
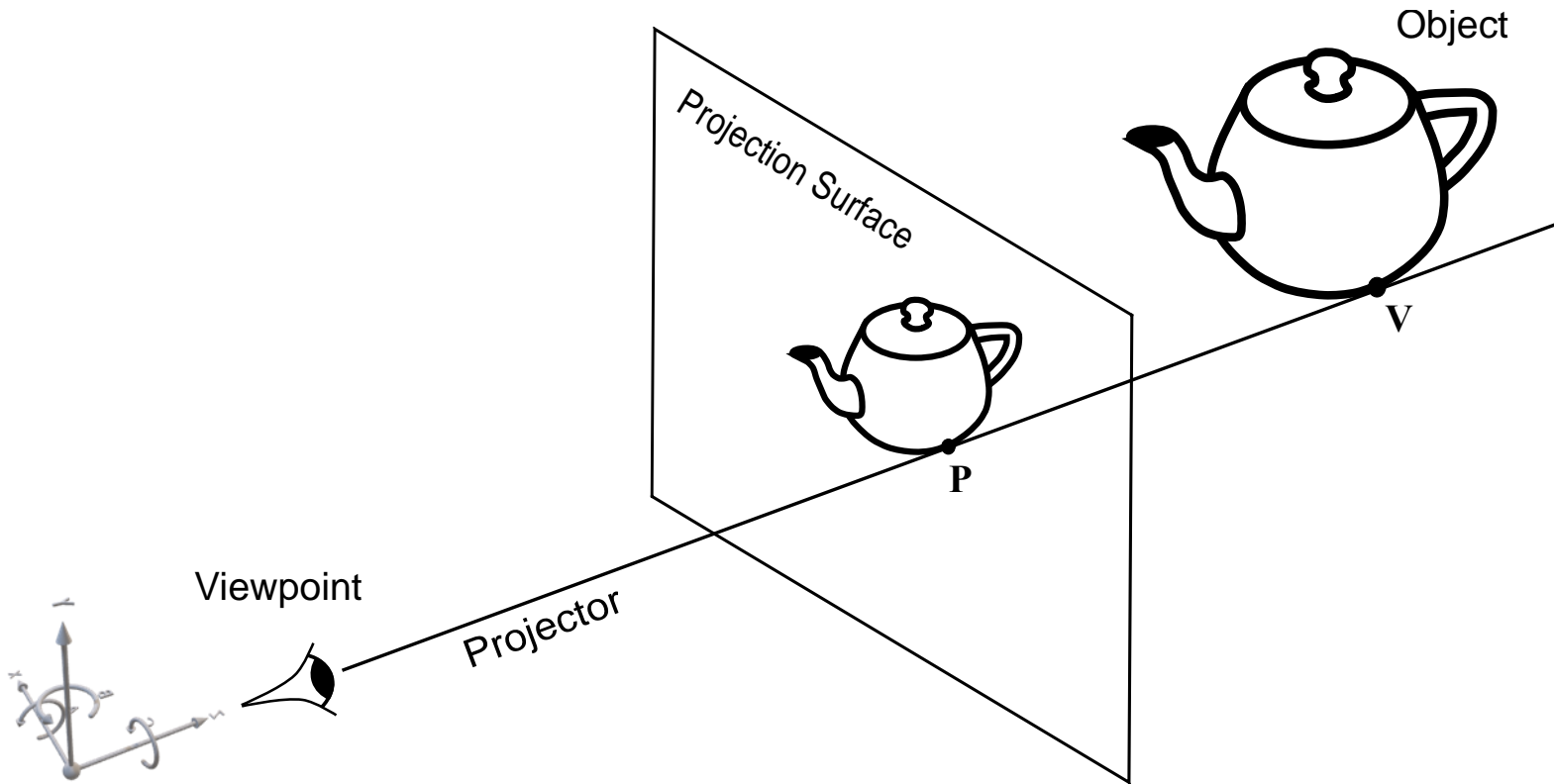
The projector takes a point on the object to a point on 2D projection surface.
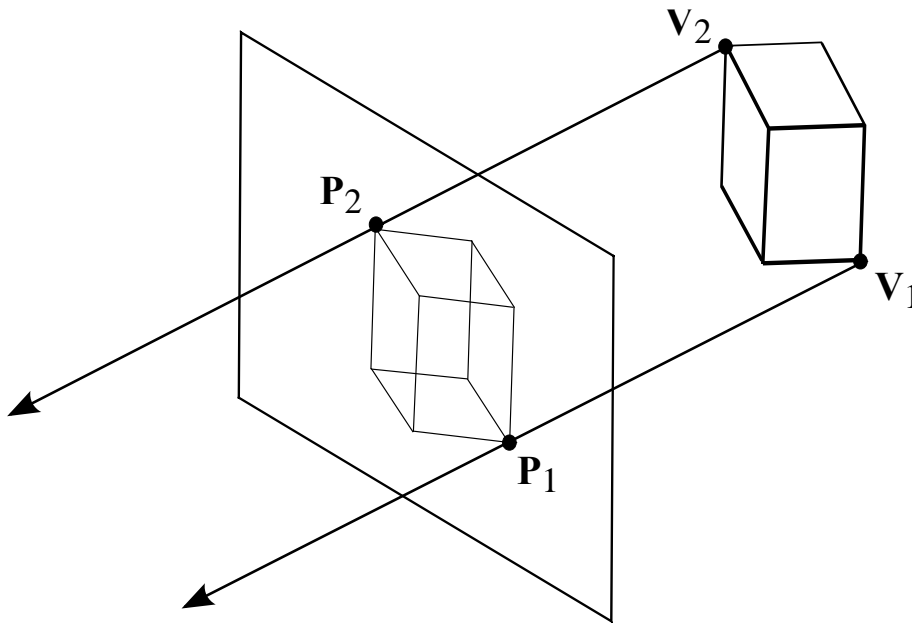
Non-linear projections

- In general it is possible to project onto any surface:
  - Sphere
  - Cone
  - Etc.

- or to use curved projectors, for example to produce lens effects.

- But we will only consider linear projections onto a flat (planar) surface.

Orthographic projection

- This is the simplest form of projection, and effective in many cases.

- Make simplifying assumptions:
  - The viewpoint is at $z = -\infty$
  - The plane of projection is $z = 0$

- So all projectors have the same direction:

$$\mathbf{d} = \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix}$$

Orthographic projection onto z = 0

Each projection line has equation

$$\mathbf{P} = \mathbf{V} + \mu\,\mathbf{d}$$

where

$$\mathbf{d} = \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix}$$

Calculating an orthographic projection

- Substitute $\mathbf{d} = (0, 0, -1)^{\mathrm{T}}$ into the projector vector equation:

$$\mathbf{P} = \mathbf{V} + \mu\mathbf{d}$$

- Gives Cartesian equations for each component

$$P_x = V_x + 0 \qquad P_y = V_y + 0 \qquad P_z = V_z - \mu$$

- Projection plane is $z = 0 \Rightarrow P_z = 0$

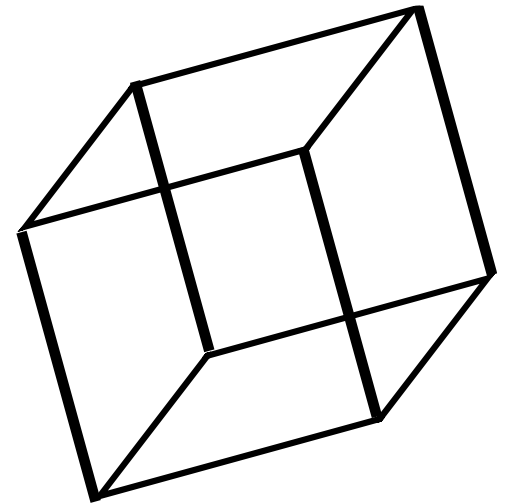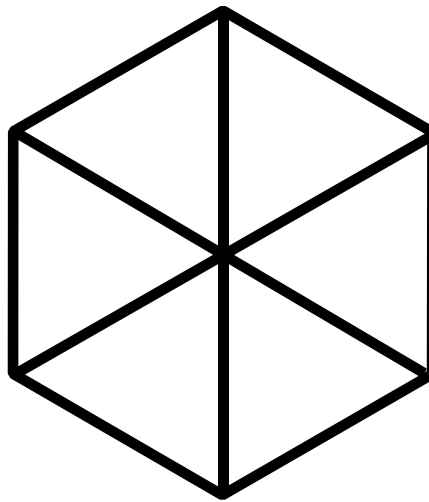# *Calculating an orthographic projection (cont.)*

- So the projected location on the screen is

$$\mathbf{P} = \begin{pmatrix} Vx \\ Vy \\ 0 \end{pmatrix}$$

- i.e. we simply take the 3D x and y components of the vertex!

Orthographic projections of a cube

- Looking at a face, a vertex and a more general view…
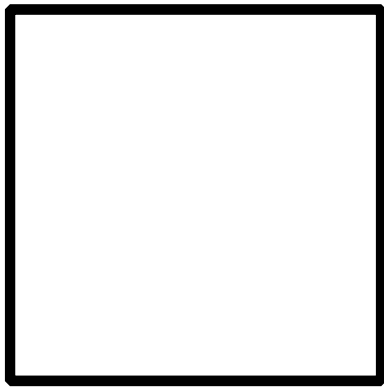
Perspective projection

- Orthographic projection is fine in cases where we are not worried about depth

    - e.g. when most objects are at the same distance from the viewer

- However, for close work - particularly computer games - it will not do.

- Instead, we use *perspective projection*.

Canonical form for perspective projection

$\mathbf{V}_1$

$\mathbf{V}_2$

y

z

$\mathbf{P}_1$

$\mathbf{P}_2$

Projection plane: $z = f$

Viewpoint

x

f
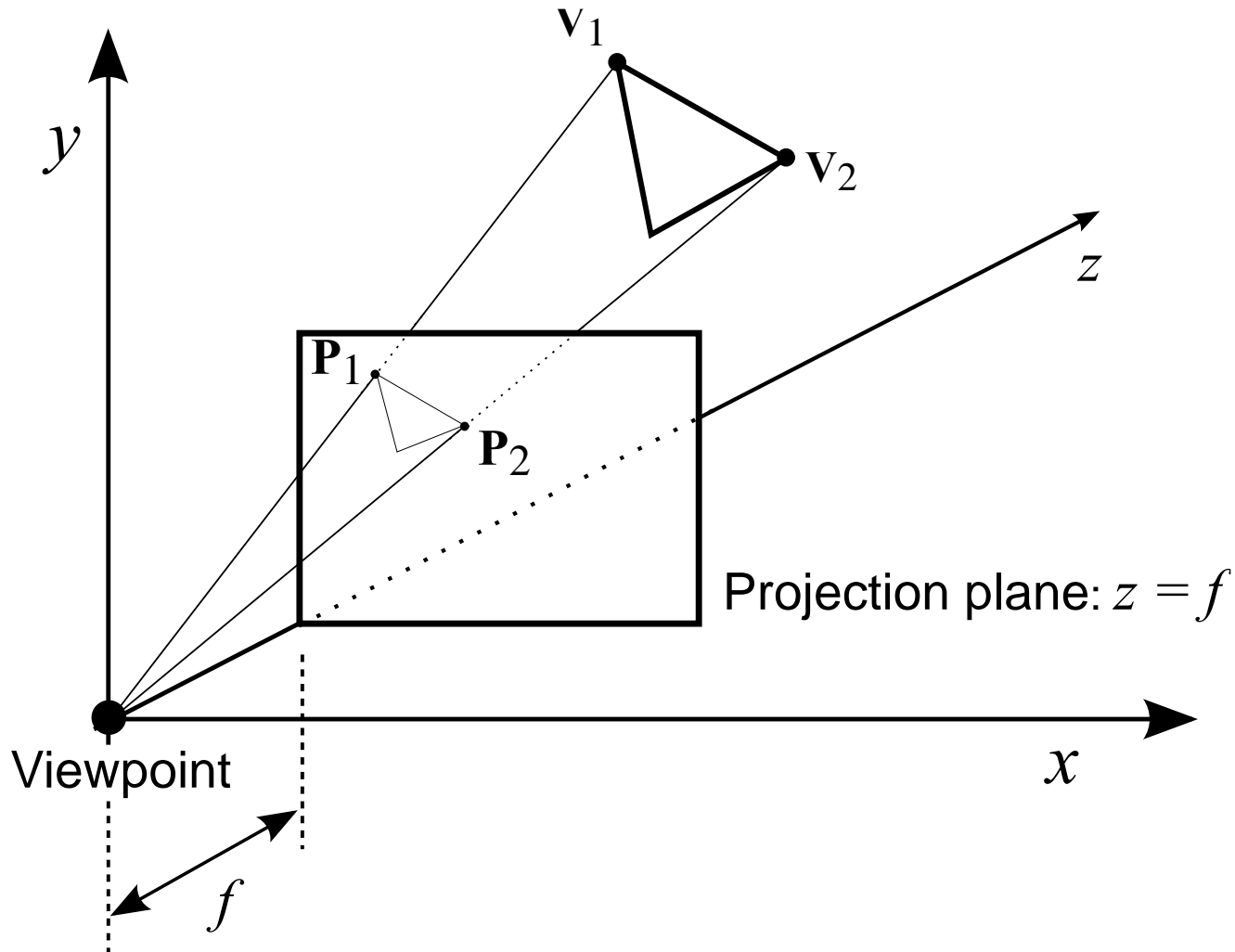
Calculating perspective projection

The perspective projector equation from vertex $\mathbf{V}$ is

$$\mathbf{P} = \mu \mathbf{V}$$

because all projectors go through the origin. At the projected point we have $P_z = f$.

Let the value of μ at this point be μ_p

$$\mu_p = P_z/V_z = f/V_z$$

and

$$P_x = \mu_p V_x, \quad P_y = \mu_p V_y$$

Therefore

$$P_x = fV_x/V_z, \quad P_y = fV_y/V_z$$

Perspective projections of a cube

a
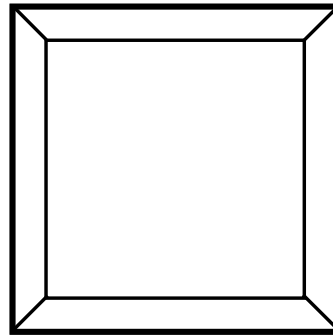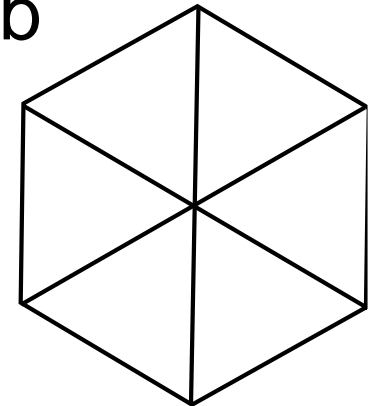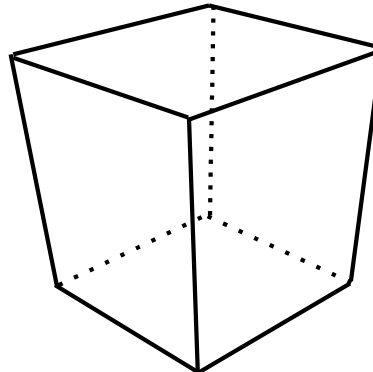
(a) Viewing a face

(b) Viewing a vertex

(c) A general view

b

c

Problem break

Given that the viewing plane is at $z = 5$, what point on the view plane corresponds to the 3D vertex

$$\mathbf{V} = \begin{pmatrix} 10 \\ 10 \\ 10 \end{pmatrix}$$

when we use the different projections:

    1. Perspective

    2. Orthographic

Problem break

Given that the viewing plane is at $z = 5$, what point on the view plane corresponds to the 3D vertex

$$\mathbf{V} = \begin{pmatrix} 10 \\ 10 \\ 10 \end{pmatrix}$$

when we use the different projections:

1. Perspective $P_x = fV_x/V_z = 5$ and $P_y = fV_y/V_z = 5$
2. Orthographic $P_x = 10$ and $P_y = 10$

The need for transformations

- Graphics scenes are defined in a particular coordinate system.

- We want to draw a graphics scene from any angle

- **But** to draw a graphics scene, it is a lot easier to have:
  - The viewpoint at the origin
  - The z-zaxis as the direction of view

- Hence, we need to be able to transform the coordinates of a graphics scene.

Transformation of viewpoint
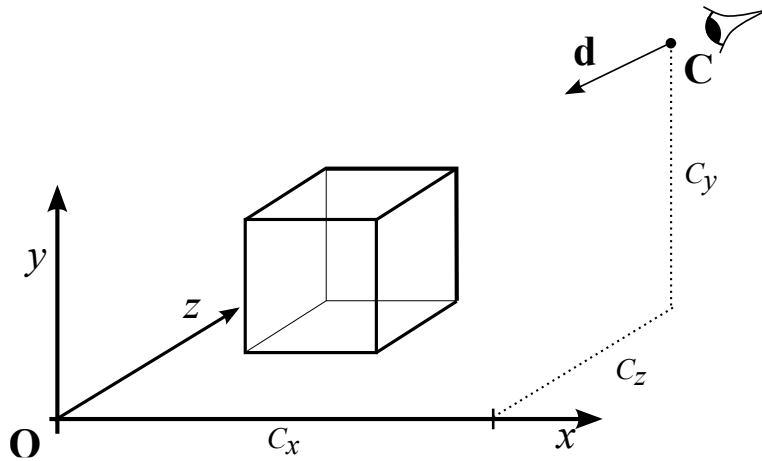
Before transformation
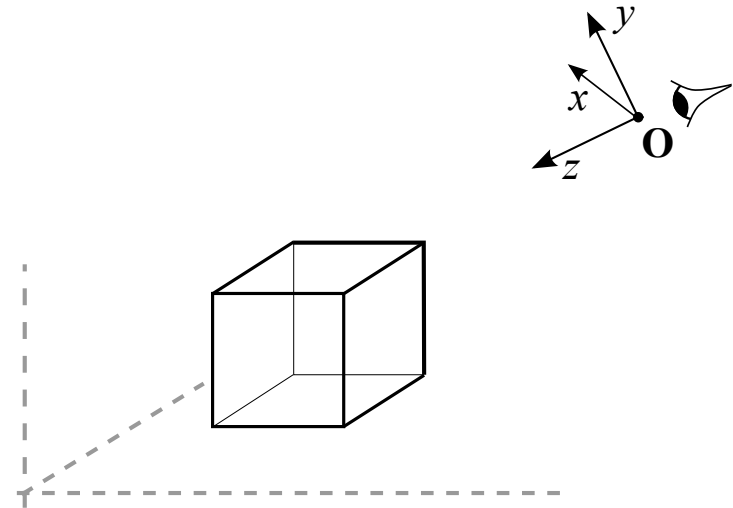
After transformation

Other transformations

- We also need transformations for other purposes:
  - Animating Objects
    e.g. flying titles, rotating, shrinking etc.
  - Multiple Instances
    the same object may appear at different places or different
    sizes
  - Reflections and other special effects

Matrix transformations of points

To transform points we use matrix multiplications, e.g. to make an object at the origin twice as big we could use:

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

which, when multiplied out, gives:

$$x' = 2x \quad y' = 2y \quad z' = 2z$$

Translation by matrix multiplication

- Many of our transformations will require translation of the points. For example if we want to move all the points two units along the x-axis we would require

$$
\begin{aligned}
x' &= x + 2 \\
y' &= y \\
z' &= z
\end{aligned}
$$

- But how can we do this with a matrix? I.e.

$$
\begin{pmatrix} & ? & \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x + 2 \\ y \\ z \end{pmatrix}
$$

… can't be done

Homogenous coordinates

- The answer is to use 4D homogenous coordinates.
- They have a 4th ordinate allowing us to use the last column for translation

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

- which, when multiplied out, gives:

$$x' = x + 2 \quad y' = y \quad z' = z$$

General homogenous coordinates

- In most cases the last ordinate will be 1

- But in general, it is a scale factor.

Homogeneous   Cartesian

$$(p_x, p_y, p_z, s) \quad \Longleftrightarrow \quad \left(\frac{p_x}{s}, \frac{p_y}{s}, \frac{p_z}{s} \right)$$

Affine transformations

- Affine transformations are those that preserve parallel lines.

- Most transformations we require are affine, the most important being:
  - Scaling
  - Rotation
  - Translation

- Other more complex transforms can be built from these.

- An example of a non-affine transformation:
  - Perspective projection (parallels not preserved).

Translation with a matrix

- We can apply a general translation by (t_x, t_y, t_z) to the points of a scene by using the following matrix multiplication

$$
\begin{pmatrix}
1 & 0 & 0 & t_x \\
0 & 1 & 0 & t_y \\
0 & 0 & 1 & t_z \\
0 & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
p_x \\
p_y \\
p_z \\
1
\end{pmatrix}
=
\begin{pmatrix}
p_x + t_x \\
p_y + t_y \\
p_z + t_z \\
1
\end{pmatrix}
$$

Inverting a translation

- Since we know what a translation matrix physically does, we can write down its inversion directly, e.g.

Translation matrix   inverse

$$\begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & -t_x \\ 0 & 1 & 0 & -t_y \\ 0 & 0 & 1 & -t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Can you show that the product of these matrices is the identity?

Scaling with a matrix

- Scaling simply multiplies each ordinate by a scaling factor.

- It can be done with the following homogenous matrix:

$$\begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} = \begin{pmatrix} s_x p_x \\ s_y p_y \\ s_z p_z \\ 1 \end{pmatrix}$$

Inverting a scaling

- To invert a scaling we simply divide the individual ordinates by the scale factor.

Scaling matrix   inverse

$$
\begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}
\begin{pmatrix} 1/s_x & 0 & 0 & 0 \\ 0 & 1/s_y & 0 & 0 \\ 0 & 0 & 1/s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}
$$

Combining transformations

- Suppose we want to make an object centred at the origin twice as big and then move it so that the centre is at $(5, 5, 20)$.

- The transformation is a scaling followed by a translation:

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 5 \\ 0 & 1 & 0 & 5 \\ 0 & 0 & 1 & 20 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Combined transformations

- We can multiply out the transformation matrices

- This gives us a single matrix which we can use to apply both transformations to any point

$$
\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 2 & 0 & 0 & 5 \\ 0 & 2 & 0 & 5 \\ 0 & 0 & 2 & 20 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}
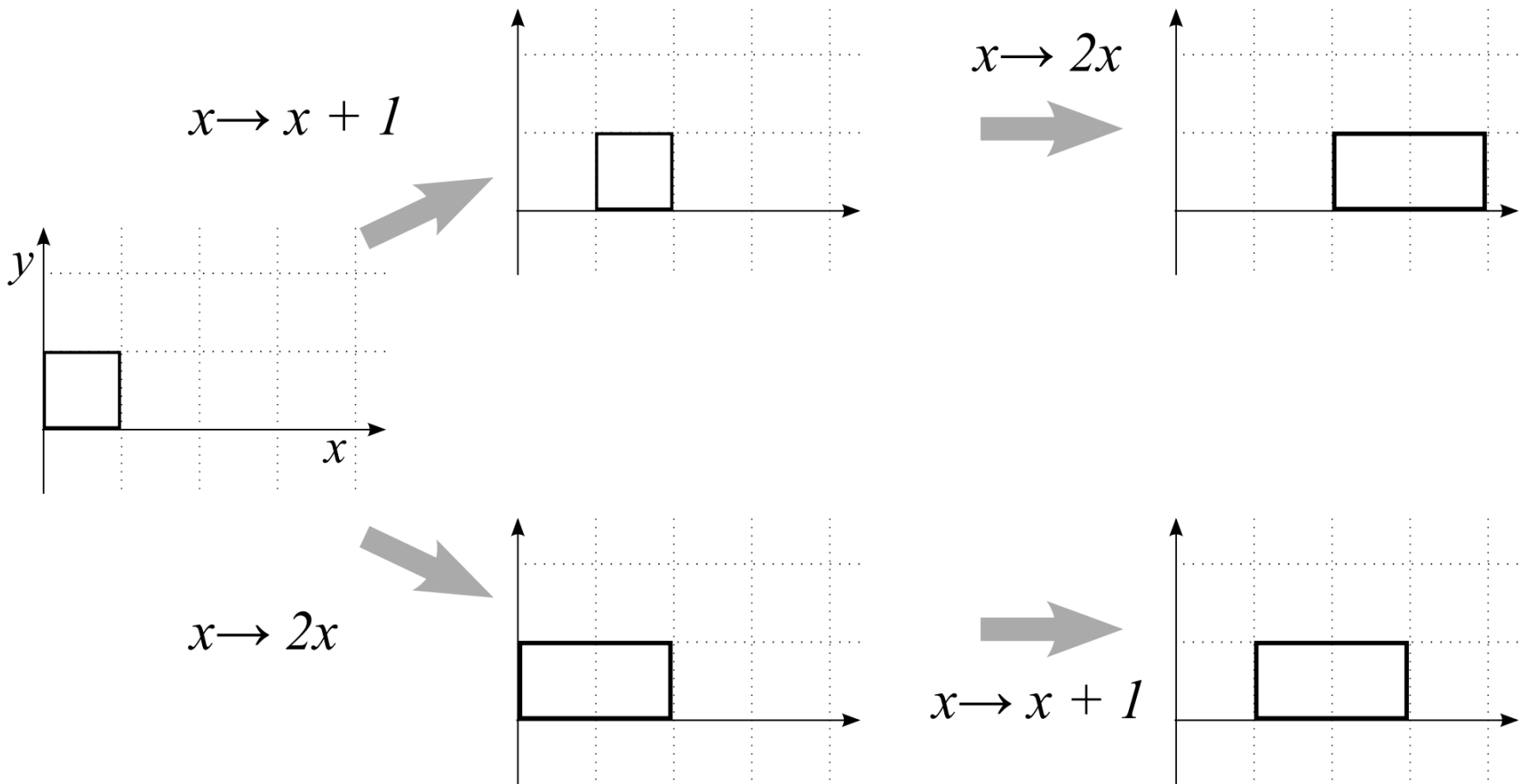$$

Careful: Transformations are not commutative

- The order of applying transformations matters:

- In general

$$\mathbf{T} \bullet \mathbf{S} \text{ is not the same as } \mathbf{S} \bullet \mathbf{T}$$

- *Check this for the transformation matrices on the last two slides*

The order of transformations is significant

The results at the end of each route are different.

Rotation

- To define a rotation, we need an axis and an angle.

- The simplest rotations are about the Cartesian axes.

- For example:

  - R_x Rotate about the x-axis
  - R_y Rotate about the y-axis
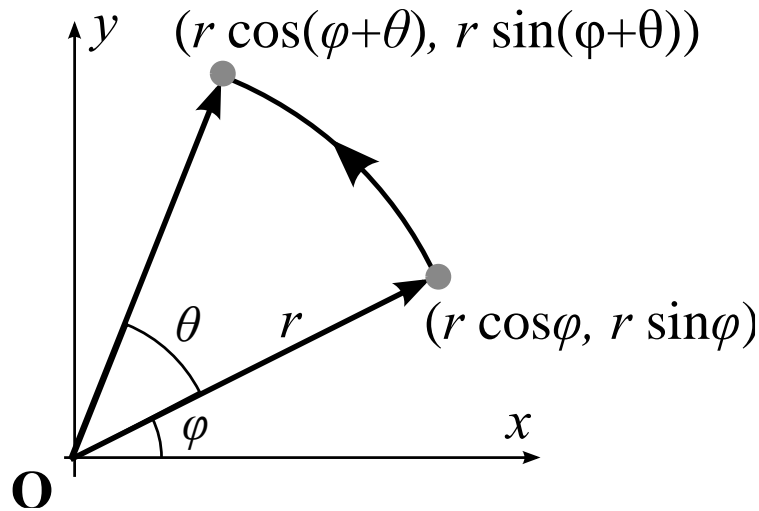 - R_z Rotate about the z-axis

Rotation matrices

By θ about each of the axes

$$\mathcal{R}_x = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\mathcal{R}_y = \begin{pmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\mathcal{R}_z = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Example: Derivation of $\mathcal{R}_z$

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} r\cos\varphi \\ r\sin\varphi \end{pmatrix}$$

$$\rightarrow \begin{pmatrix} r\cos(\varphi+\theta) \\ r\sin(\varphi+\theta) \end{pmatrix}$$

$$= \begin{pmatrix} r\cos\varphi\cos\theta - r\sin\varphi\sin\theta \\ r\cos\varphi\sin\theta + r\sin\varphi\cos\theta \end{pmatrix}$$

$$= \begin{pmatrix} x\cos\theta - y\sin\theta \\ x\sin\theta + y\cos\theta \end{pmatrix}$$

$$= \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

$$\downarrow \qquad \downarrow$$

$$\begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

y $(r\cos(\varphi+\theta),\ r\sin(\varphi+\theta))$

$(r\cos\varphi,\ r\sin\varphi)$

θ r

φ

x

O

z-axis goes into page

Rotations have a direction

- Note the following about the matrix formulations given in these notes:
  - We will stick to a left-handed coordinate system
  - Rotation is anti-clockwise when looking along the axis of rotation (in the previous slide, the z-axis goes into the page).
  - Rotation is clockwise when looking back towards the origin from the positive side of the axis

y

z

x
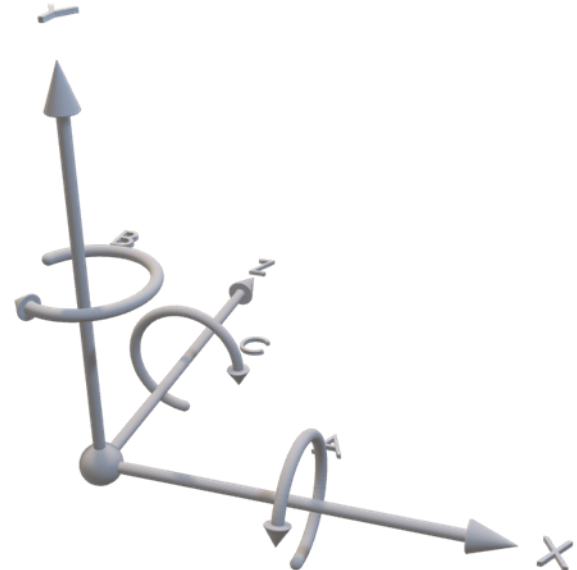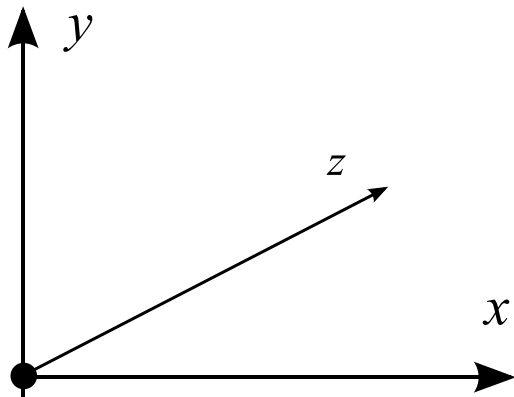
Inverting rotation

Inverting a rotation
by angle θ

$\Leftrightarrow$

Rotating through
angle $-\theta$

• i.e. we can use the following relations to help us find the inverse of a rotation:

$$\cos(-\theta) = \cos(\theta) \qquad \text{and} \qquad \sin(-\theta) = -\sin(\theta)$$

Inverting rotation

- So for example:

Rotation

Inverse

$$\mathcal{R}_z(\theta)$$

$$\mathcal{R}_z(-\theta)$$

$$\begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad \begin{pmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$