Revision

Lecturer: Rachel McDonnell
Assistant Professor in Creative Technologies
Rachel.McDonnell@cs.tcd.ie

Course www: Blackboard

Note: answers in these lecture slides are not exact solutions, but revision of the lecture notes that relate to the topics.. The answers will have been discussed in class

Student Survey

- Please take 5 minutes to complete the online anonymous student survey to provide feedback on the course

https://goo.gl/forms/szKtG4pOCmkyDrZG3

Structure

UNIVERSITY OF DUBLIN

TRINITY COLLEGE

Faculty of Engineering, Mathematics and Science

School of Computer Science and Statistics

Annual Examination 2014

Computer Science

Fourth Year Examination

CS4052 – Computer Graphics

2014 **Venue** **time**

Dr. Rachel McDonnell

Instructions to Candidates:

Answer any FOUR questions – 25 marks each.
All questions carry equal marks.

Please use a **separate answer book** for each question.
The entire question paper must be handed in at the end of the examination.

What you should revise

- Go over all the lecture topics
- Major themes that we covered
  - Hardware pipeline and shaders
  - Linear algebra and geometric problems (dot and cross products)
  - Transformations
  - Hierarchies & rotations
  - Viewing
  - Illumination
  - Ray-tracing
  - Animation (High-level Questions)
  - Splines (not covered this year)
  - Mapping (High-level Questions)

Intended grading scheme

- significant gaps in knowledge < 50%
- basic working knowledge of computer graphics – 50%
- can solve some variations to common problems – 60-70%
- broad knowledge of basic theory and practice – 80%
- has also thought about side-topics, has more advanced theory – up to 100%

Potential Problems

- Don’t know the answer
  - Write down what you know about the topic
- Mental block / forgot a term
  - draw a diagram and add description to explain
- “This question doesn't make sense!”
  - ask (I will be available at the start of the exam)

Level of Detail?

- High-level questions
  - Mapping
  - Animation (including splines)

Mapping/Animation Question

- You have been given the following character model (created in a modelling package) that you would like to use in your real-time game.
    1. discuss 3 texture mapping techniques you would use to get this character looking more realistic
    2. How would you prepare the character for animation?
- Name and discuss 3 different techniques you would use to animate the character.

Possible Solution

- Diffuse texture map for the skin and clothing colours
- Normal map for the clothing details
- Displacement map for the larger wrinkles and muscle details

Possible Solution

- Create hierarchical skeleton
- Use rigging to associate the mesh with the skeleton
- Weight each bone's influence on each vertex

Possible Solution

- Physically based animation for hair
- Cloth simulation for the clothing
- Inverse kinematics for the sword or foot-placement
- Motion capture
- Keyframing

Level of Detail?

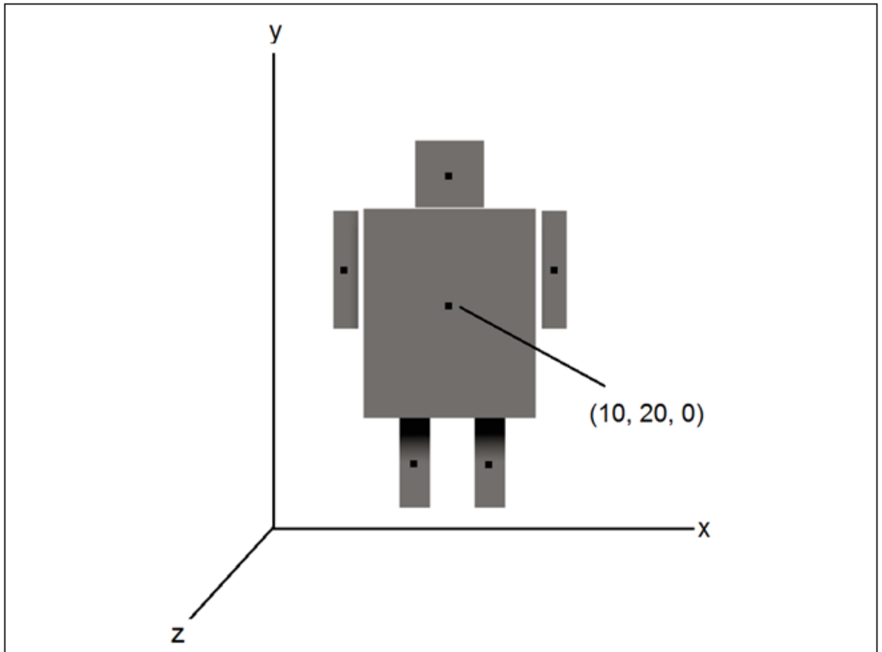
- Detailed questions on:
  - Linear algebra
    - Cross/Dot product
    - etc.
  - Graphics programming
    - write a shader, what does this shader do?
    - know the pipeline, etc.
  - Geometry
    - Given this model, how would you animate?
    - Rotation of object about arbitrary axis, etc
  - Viewing
    - derive perspective matrix
    - given this view, how would you produce it?

Level of Detail?

- Detailed questions on:
  - Illumination
    - Phong illumination model in detail
    - Phong/Gouraud shading
  - Ray tracing
    - ray intersection tests
    - pseudo-code for algorithm
    - speed-ups

Question 1

Question 1: Geometric Transformations



Roughly estimate the positions of the various joints depicted in the hierarchy above and provide the modern shader-based OpenGL code necessary to create this object. Assume that we have already loaded a vertex buffer object with the necessary vertex data for the box object, and have enabled and bound that buffer. Also, assume that you have access to functions to rotate, scale, and translate 4 x 4 matrices. (10 marks) i. Now write a function to scale a 4 x 4 matrix in the x-, y-, and z-axes (5 marks) ii. How would you move the whole assembly to the right by 5 units? (3 marks) iii. Describe how to create a walk cycle for the character in the forward direction (using a Right handed system). Its hands and legs should move appropriately. (7 marks)	(25 marks)

Relative Motion

- Interested in animating objects whose motion is relative to another object
- Such a sequence is called a *motion hierarchy*
- Components of a hierarchy represent objects that are physically connected or *linked*
- In some cases, motion can be restricted
  - Reduced dimensionality
  - Hierarchy enforces constraints
- Two approaches for animating figures defined by hierarchies: forward & inverse kinematics

OpenGL® Implementation

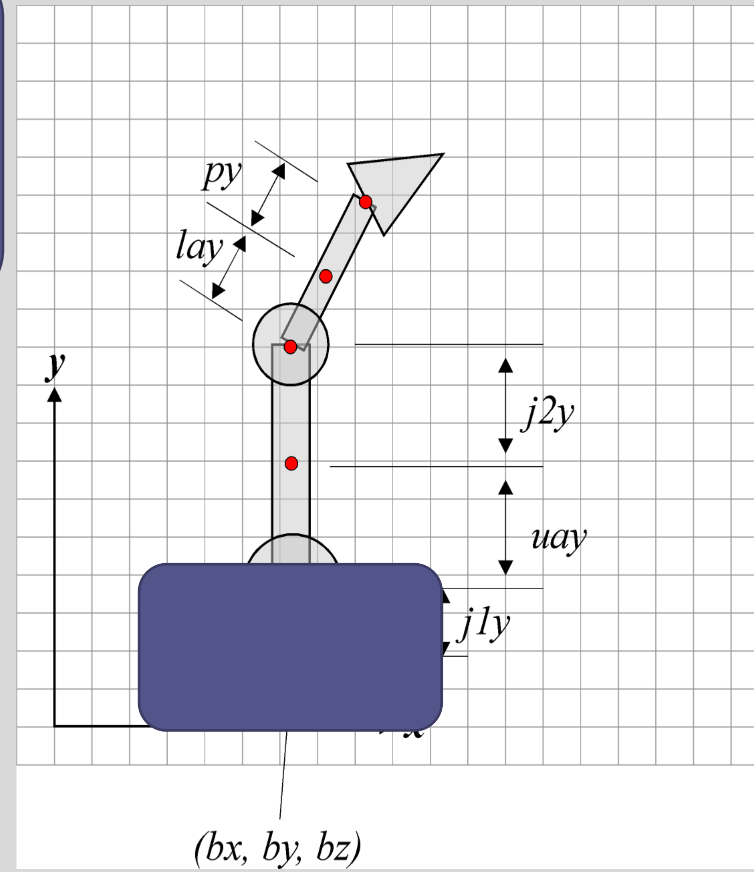
```
local2 = identity_mat4 ();
local2 = rotate(joint1_orientation) * local2;
local2 = translate(0, j1y, 0) * local2;
global2 = local1*local2;

updateUniformVariables(model matrix = global2);
drawJoint1();

local3 = identity_mat4 ();
local3 = rotate(upperArm_orientation) * local3;
local3 = translate(0, uay, 0) * local3;
global3 = local1*local2*local3;

updateUniformVariables(model matrix = global3);
drawUpperArm();

etc.
```



OpenGL® Implementation

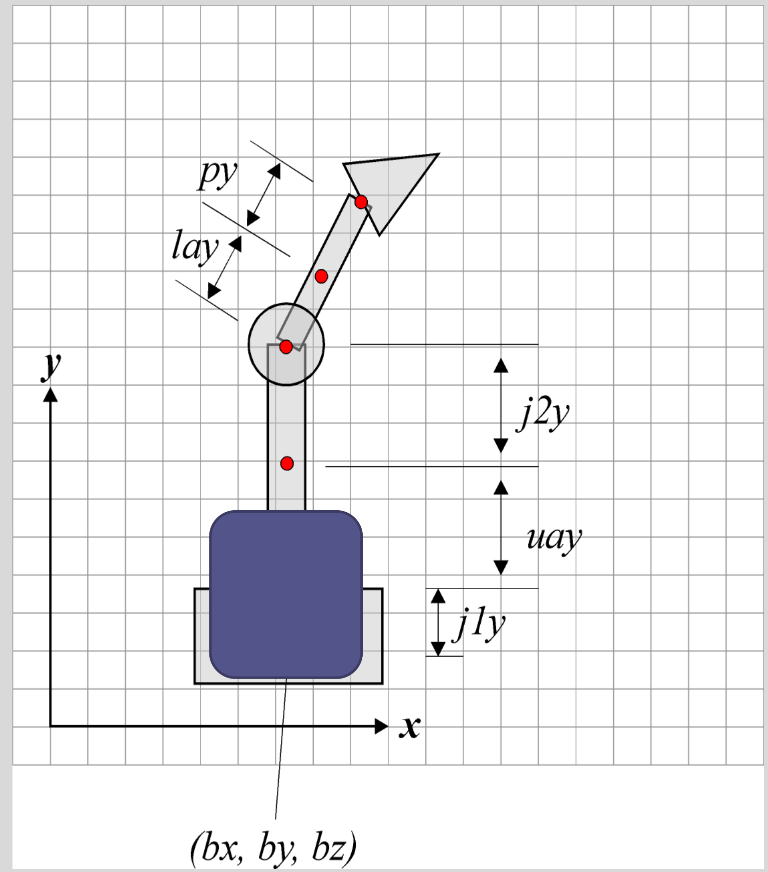
```
local1 = identity_mat4 ();
local1 = rotate(base_orientation) * local1;
local1 = translate(bx, by, bz) * local1;
global1 = local1;

updateUniformVariables(model matrix = global1);
drawBase();
```

```
local3 = identity_mat4 ();
local3 = rotate(upperArm_orientation) * local3;
local3 = translate(0, uay, 0) * local3;
global3 = local1*local2*local3;

updateUniformVariables(model matrix = global3);
drawUpperArm();

etc.
```



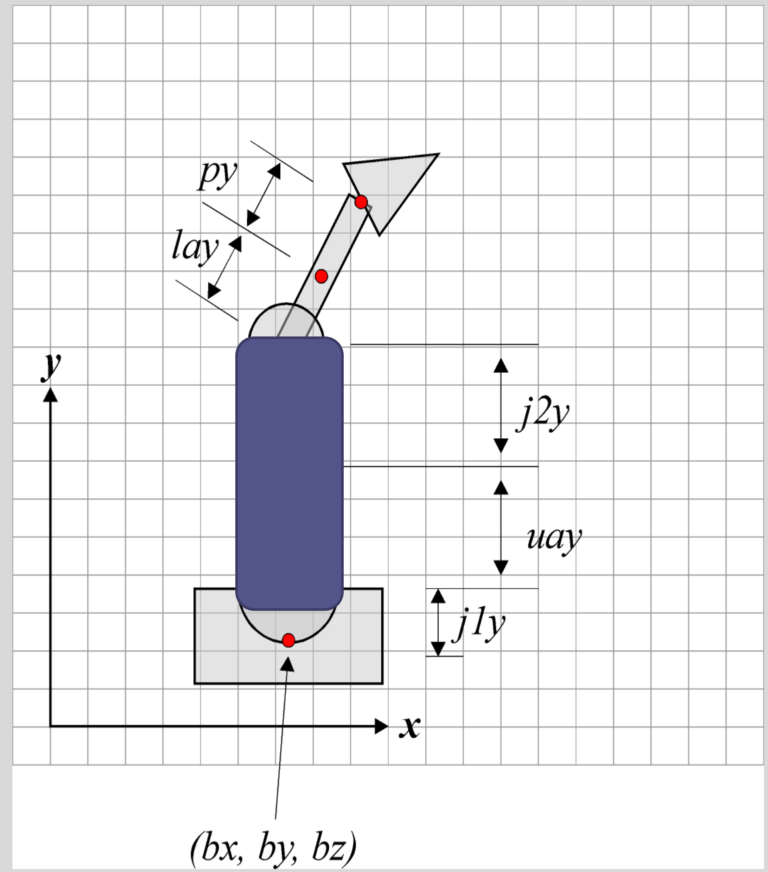
OpenGL® Implementation

```
local1 = identity_mat4 ();
local1 = rotate(base_orientation) * local1;
local1 = translate(bx, by, bz) * local1;
global1 = local1;

updateUniformVariables(model matrix = global1);
drawBase();

local2 = identity_mat4 ();
local2 = rotate(joint1_orientation) * local2;
local2 = translate(0, j1y, 0) * local2;
global2 = local1*local2;

updateUniformVariables(model matrix = global2);
drawJoint1();
```



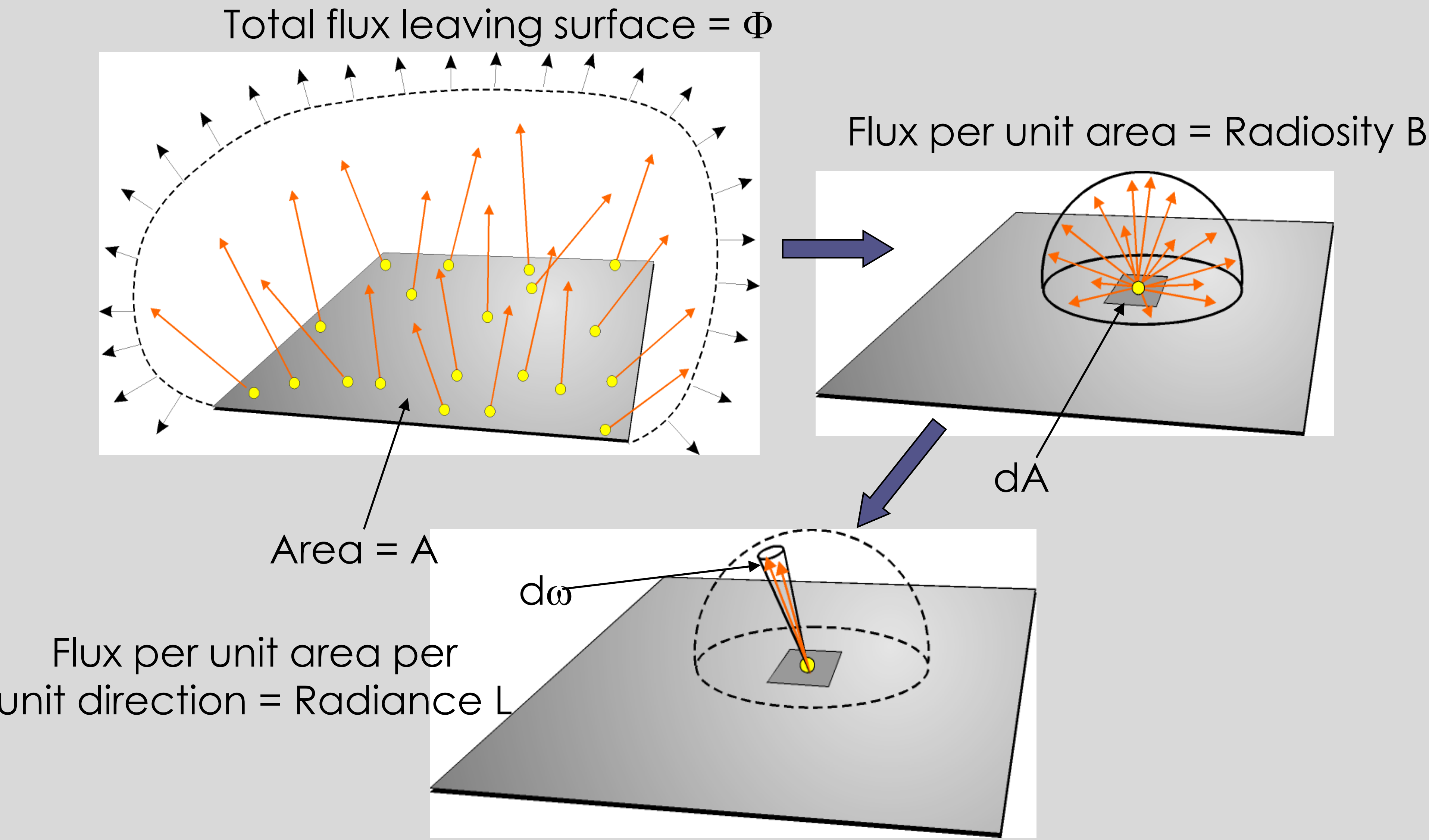

```
etc.
```

Question 2

Question 2: Illumination

(a)	Explain the following terms: i. Flux ii. Radiosity iii. Radiance	(5 Marks)
(b)	Explain the difference between local and global illumination of a point.	(5 Marks)
(c)	Explain each of the following with respect to illumination models (using diagrams and equations where appropriate): i. The Inverse Square Law ii. The Cosine Rule iii. BRDF (include examples)	(15 Marks)

Radiometric Units



Question 2

Question 2: Illumination

(a)	Explain the following terms: i. Flux ii. Radiosity iii. Radiance	(5 Marks)
(b)	Explain the difference between local and global illumination of a point.	(5 Marks)
(c)	Explain each of the following with respect to illumination models (using diagrams and equations where appropriate): i. The Inverse Square Law ii. The Cosine Rule iii. BRDF (include examples)	(15 Marks)

Rendering Algorithms

- Rendering algorithms differ in the assumptions made regarding lighting and reflectance in the scene and in the solution space:
  - **local illumination** algorithms: consider lighting only from the light sources and ignore the effects of other objects in the scene (i.e. reflection off other objects or shadowing)
  - **global illumination** algorithms: account for all modes of *light transport*
  - **view dependent** solutions: determine an image by solving the illumination that arrives through the viewport only.
  - **view independent** solutions: determine the lighting distribution in an entire scene regardless of viewing position. Views are then taken after lighting simulation by sampling the full solution to determine the view through the viewport.

Question 2

Question 2: Illumination

(a)	Explain the following terms: i. Flux ii. Radiosity iii. Radiance	(5 Marks)
(b)	Explain the difference between local and global illumination of a point.	(5 Marks)
(c)	Explain each of the following with respect to illumination models (using diagrams and equations where appropriate): i. The Inverse Square Law ii. The Cosine Rule iii. BRDF (include examples)	(15 Marks)

Normalised Vectors

- When we wish to describe direction we use *normalised* vectors.
- We normalise a vector by dividing by its magnitude:

$$\mathbf{v}' = \frac{\mathbf{v}}{\|\mathbf{v}\|} = \frac{1}{\sqrt{v_1^2 + v_2^2 + \cdots + v_n^2}}\mathbf{v}$$

Dot Product

- Dot product (*inner product*) is defined as:

$$\mathbf{u} \cdot \mathbf{v} = \sum_i u_i v_i$$

$$\mathbf{u} \cdot \mathbf{v} = \mathbf{u}^T \mathbf{v} = \begin{bmatrix} u_1 & u_2 & u_3 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = u_1 v_1 + u_2 v_2 + u_3 v_3$$

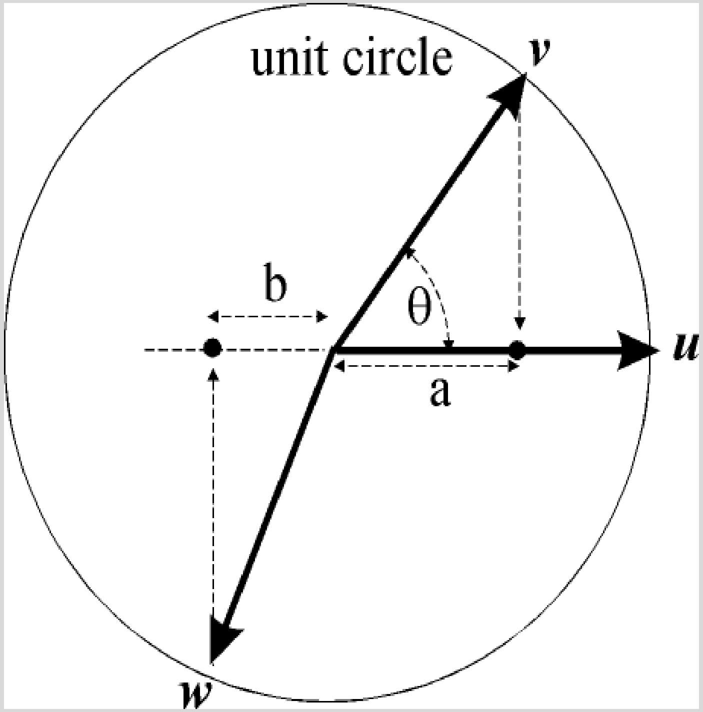
- Note: $\mathbf{u} \cdot \mathbf{u} = u_1^2 + u_2^2 + u_3^2 = \|\mathbf{u}\|^2$

- Therefore we can also define magnitude in terms of the dot-product operator: $\|\mathbf{u}\| = \sqrt{\mathbf{u} \cdot \mathbf{u}}$

- Dot product operator is *commutative*.

Dot Product

- If both vectors are normalised, the dot product defines the cosine of the angle between the vectors:



$$\mathbf{u} \cdot \mathbf{v} = \cos\theta$$

In general:

$$\mathbf{u} \cdot \mathbf{v} = \|\mathbf{u}\|\|\mathbf{v}\|\cos\theta$$

$$\Rightarrow \theta = \cos^{-1}\left[\frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\|\|\mathbf{v}\|}\right]$$

Cross Product Example

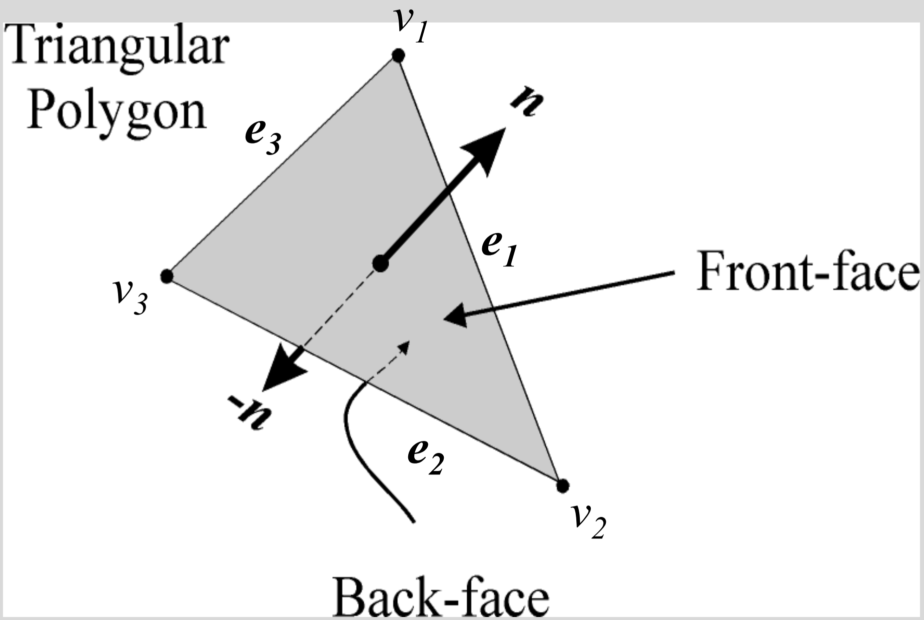
- Find **u** x **v** where **u** = (1,2,-2) and **v** = (3,0,1)

$$\mathbf{u} \times \mathbf{v} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} \times \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} u_2 v_3 - u_3 v_2 \\ u_3 v_1 - u_1 v_3 \\ u_1 v_2 - u_2 v_1 \end{bmatrix}$$

$$\mathbf{u} \times \mathbf{v} = \begin{bmatrix} 1 \\ 2 \\ -2 \end{bmatrix} \times \begin{bmatrix} 3 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 2-0 \\ -6-1 \\ 0-6 \end{bmatrix}$$

Normals & Polygons

- Polygons are (usually) planar regions bounded by *n* edges connecting *n* points or *vertices*.
- For lighting and viewing calculations we need to define the normal to a polygon:



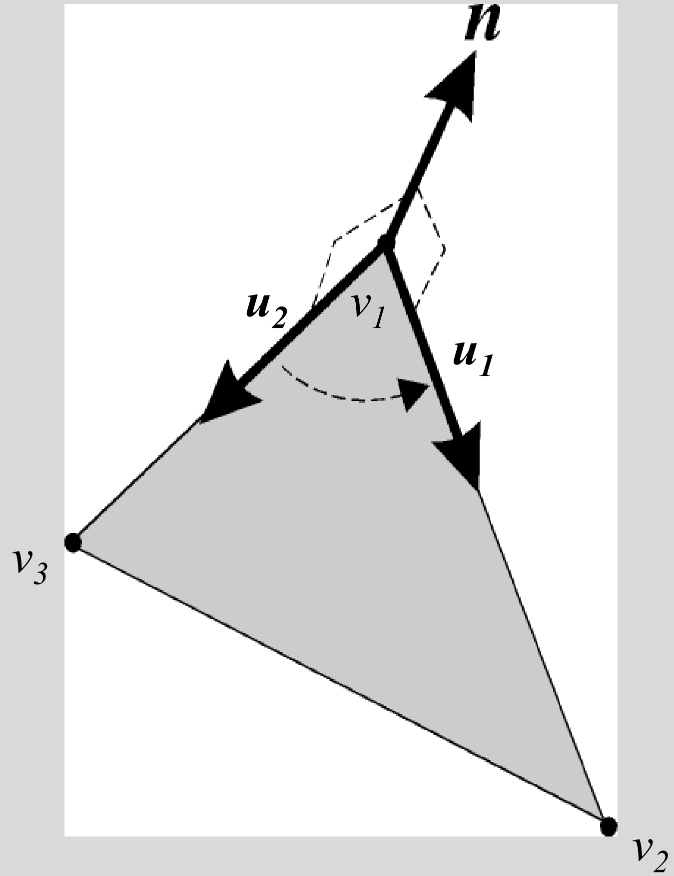
- The normal distinguishes the *front-face* from the *back-face* of the polygon.

Normals & Polygons

- First determine the 2 *edge vectors* from the vertices:

$$\mathbf{u}_1 = \frac{v_2 - v_1}{\|v_2 - v_1\|} \qquad \mathbf{u}_2 = \frac{v_3 - v_1}{\|v_3 - v_1\|}$$

- The polygon normal is given by: $$\mathbf{n} = \frac{\mathbf{u}_2 \times \mathbf{u}_1}{\|\mathbf{u}_2 \times \mathbf{u}_1\|}$$

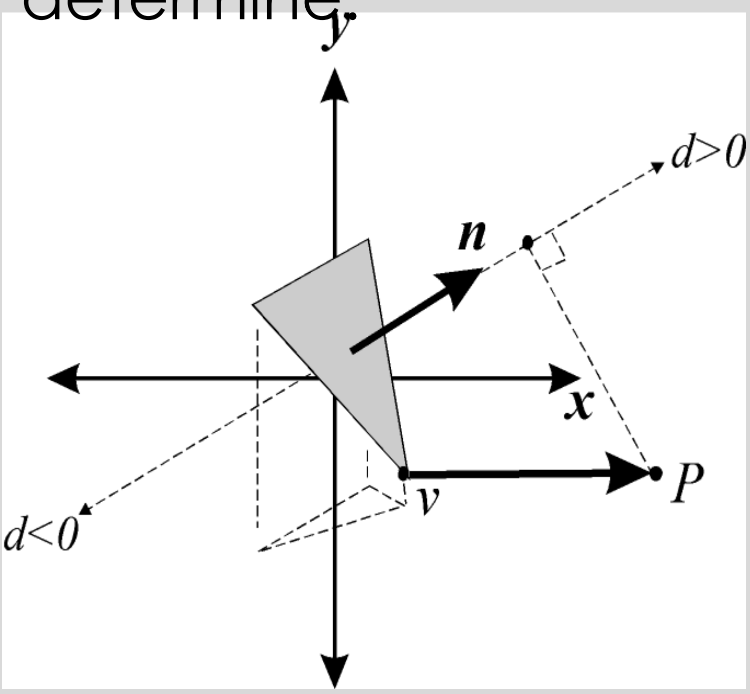


Normals & Polygons

- The plane of the polygon divides 3D space into 2 *half-spaces*
- All points *P* are either in front of or behind the polygon.
- To determine the side determine:

$$d = \mathbf{n} \cdot (P - v_i)$$

  - $d < 0 \Rightarrow P$ behind
  - $d = 0 \Rightarrow P$ on polygon
  - $d > 0 \Rightarrow P$ in front



(a) What does each of these matrices represent? ~~Explain how each is derived.~~ **(6 marks)**

i)	$\begin{bmatrix} 1/s_x & 0 & 0 & 0 \\ 0 & 1/s_y & 0 & 0 \\ 0 & 0 & 1/s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
ii)	$\begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
iii)	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix}$

Scale

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} s_x x \\ s_y y \\ s_z z \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \quad \Rightarrow \quad \mathbf{v}' = \mathbf{S}\mathbf{v}$$

We would also like to scale points thus we need a *homogeneous transformation* for consistency:

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} s_x x \\ s_y y \\ s_z z \\ w \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

$$\mathbf{S}^{-1} = \begin{bmatrix} 1/s_x & 0 & 0 & 0 \\ 0 & 1/s_y & 0 & 0 \\ 0 & 0 & 1/s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation

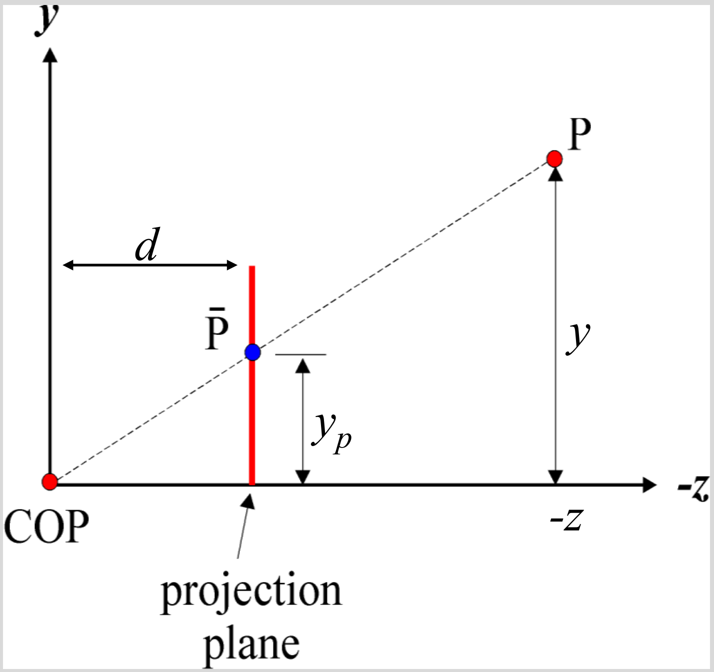
- 2D rotation of θ about origin: $\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$
- 3D homogeneous rotations:

$$\mathbf{R}_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \mathbf{R}_y = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \mathbf{R}_z = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Note: difference for rotation about y, due to RHS
- <u>Note</u>: $\begin{array}{l} \cos(-\theta) = \cos\theta \\ \sin(-\theta) = -\sin\theta \end{array} \Rightarrow \mathbf{R}^{-1}(\theta) = \mathbf{R}(-\theta) = \mathbf{R}^T(\theta)$
- If $\mathbf{M}^{-1} = \mathbf{M}^{T}$ then **M** is *orthonormal*. All orthonormal matrices are rotations about the origin.

Perspective Projections

Consider a perspective projection with the viewpoint at the origin and a viewing direction oriented along the positive -*z* axis and the view-plane located at $z = -d$



$$\frac{y}{z} = \frac{y_P}{d} \Rightarrow y_P = \frac{y}{z/d}$$

Non-uniform foreshortening

a similar construction for x_p

$\Rightarrow$

$$\begin{bmatrix} x_P \\ y_P \\ z_P \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{x}{z/d} \\ \frac{y}{z/d} \\ -d \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ -z \\ z/d \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

can modify use of homogeneous coordinates to handle projections

Transformation Matrix

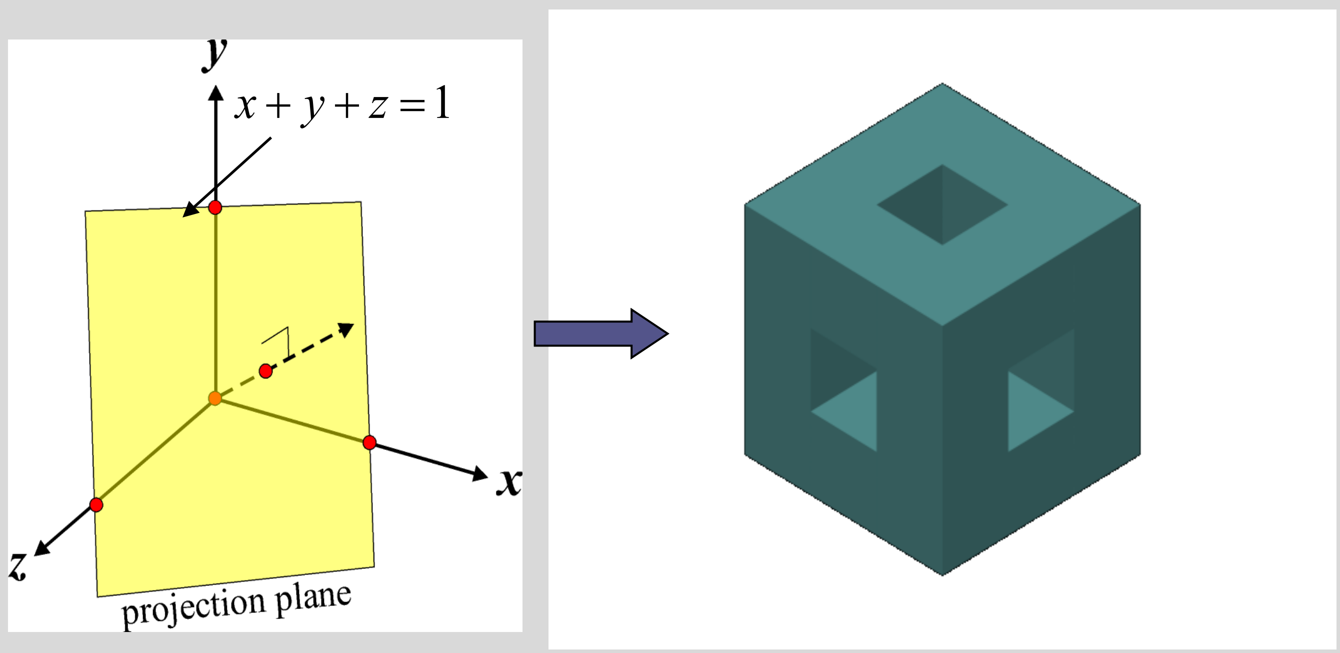
Question 4

Question 4: Viewing

(a)	Explain each of the following terms: i. Isometric projection ii. Clipping iii. Axonometric projection	(8 Marks)
(b)	What is aspect ratio? Provide the OpenGL (or similar Graphics API) code to show how a simple scene (e.g., a cube and a sphere) can be rendered using an aspect ratio of 1.25 and of 0.5, and sketch the resulting images.	(8 Marks)
(c)	Describe the process for transforming a point using a perspective projection. Use diagrams and equations where appropriate.	(9 Marks)

Orthogonal Projections

- The result is an *orthographic* projection if the object is axis aligned, otherwise it is an *axonometric* projection.
- If the projection plane intersects the principle axes at the same distance from the origin the projection is *isometric*.



Question 4

Question 4: Viewing

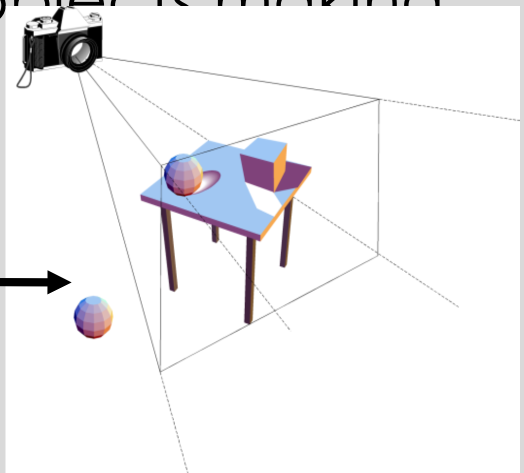
(a)	Explain each of the following terms: i. Isometric projection ii. Clipping iii. Axonometric projection	(8 Marks)
(b)	What is aspect ratio? Provide the OpenGL (or similar Graphics API) code to show how a simple scene (e.g., a cube and a sphere) can be rendered using an aspect ratio of 1.25 and of 0.5, and sketch the resulting images.	(8 Marks)
(c)	Describe the process for transforming a point using a perspective projection. Use diagrams and equations where appropriate.	(9 Marks)

Clipping

Clipping

- The computer may have model, texture, and shader data for all objects in the scene in memory
- The virtual camera viewing the scene only "sees" the objects within the field of view
- The computer does not need to transform, texture, and shade the objects that are behind or on the sides of the camera
- A clipping algorithm skips these objects making rendering more efficient

Outside view so must be clipped



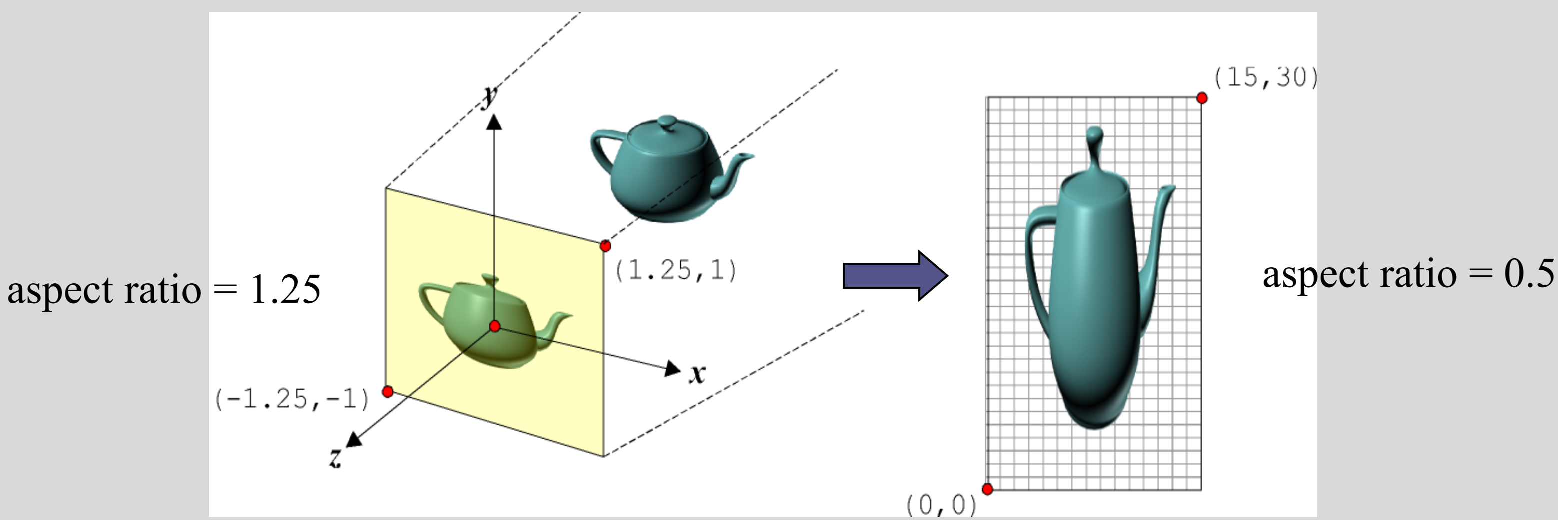
Question 4

Question 4: Viewing

(a)	Explain each of the following terms: i. Isometric projection ii. Clipping iii. Axonometric projection	(8 Marks)
(b)	What is aspect ratio? Provide the OpenGL (or similar Graphics API) code to show how a simple scene (e.g., a cube and a sphere) can be rendered using an aspect ratio of 1.25 and of 0.5, and sketch the resulting images.	(8 Marks)
(c)	Describe the process for transforming a point using a perspective projection. Use diagrams and equations where appropriate.	(9 Marks)

Aspect Ratio

- The *aspect ratio* defines the relationship between the width and height of an image.
- Using `Perspective` matrix, a viewport aspect ratio may be explicitly provided, otherwise the aspect ratio is a function of the supplied viewport width and height.
- The aspect ratio of the window (defined by the user) must match the viewport aspect ratio to prevent unwanted *affine* distortion:



Example RayTracing Question

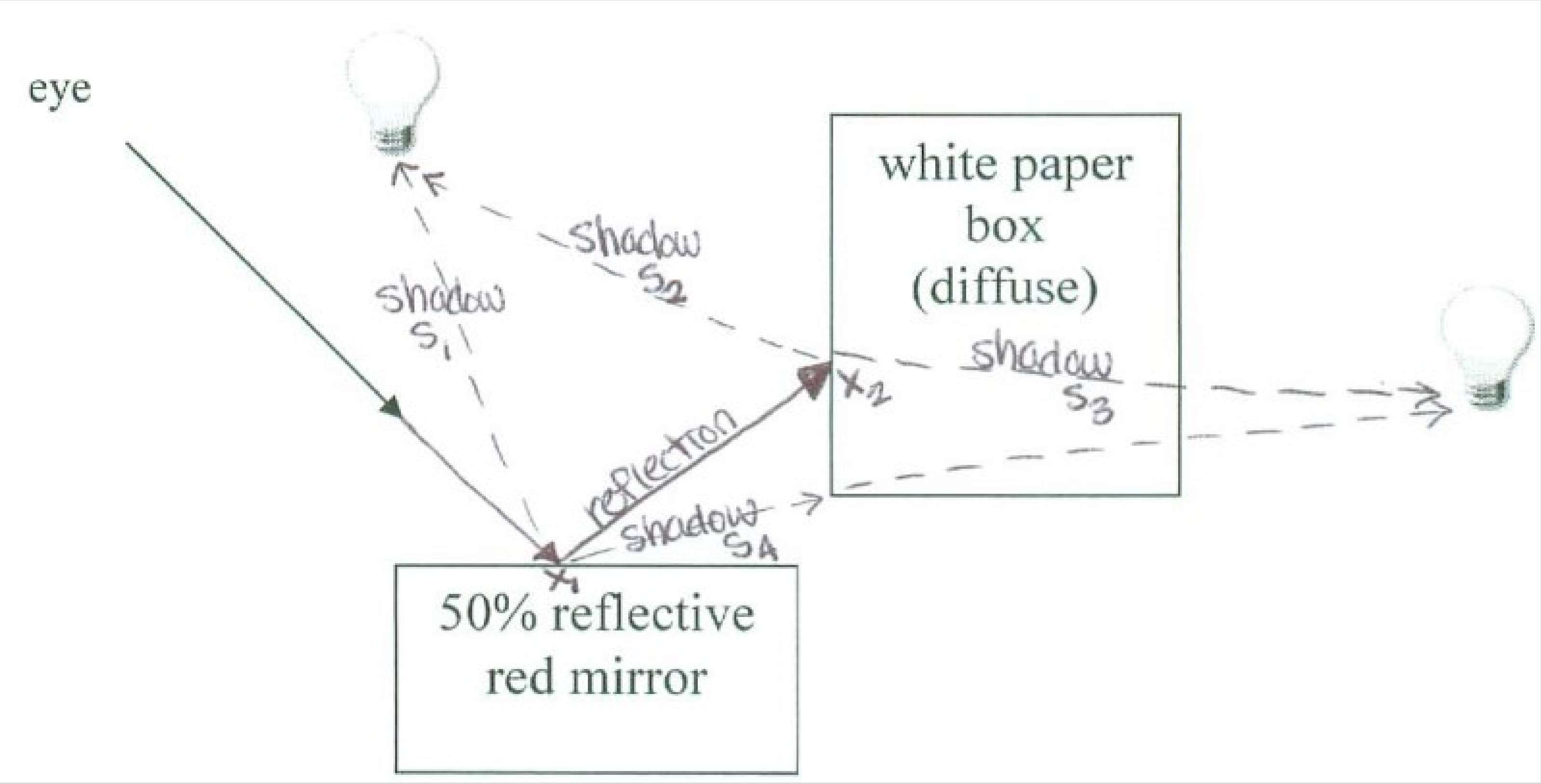
a) Provide an outline of the basic ray-tracing algorithm (You are encouraged to use pseudocode and drawings to illustrate various steps). **[11 marks]**

b) Mention a reason for why ray tracing is computationally expensive and a technique that can be used to reduce this cost. **[2 marks]**

c) There are several different factors that determine the colour of an object at a specific point. Compare local and global illumination of that point, and view dependent and view independent solutions to determining it. **[4 marks]**

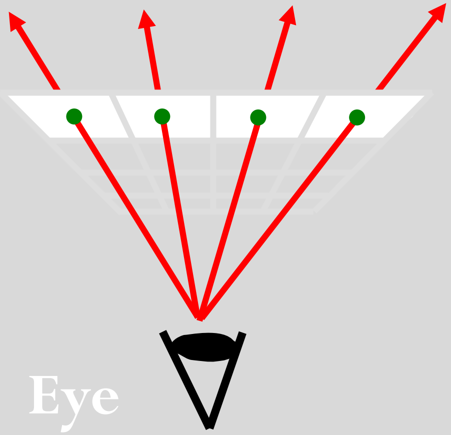
d) Outline an algorithm to find the intersection between a ray and a sphere. (Illustrate your answer with an example ray and sphere). **[8 marks]**

The following scene that we wish to ray-trace has a reflective red mirror, white paper, and two lights. Sketch the scene in your answer book and, starting from the eye ray, draw all additional reflection, refraction, and shadow rays needed to compute the colour of the eye ray.



The Ray Tracing Algorithm

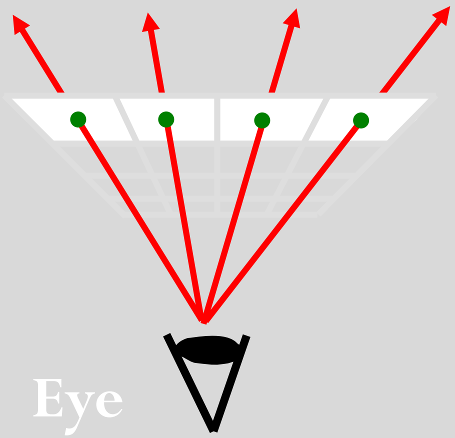
```
for each pixel in viewport
{
    determine eye ray for pixel
    intersection = trace(ray, objects)
    colour = shade(ray, intersection)
}
```

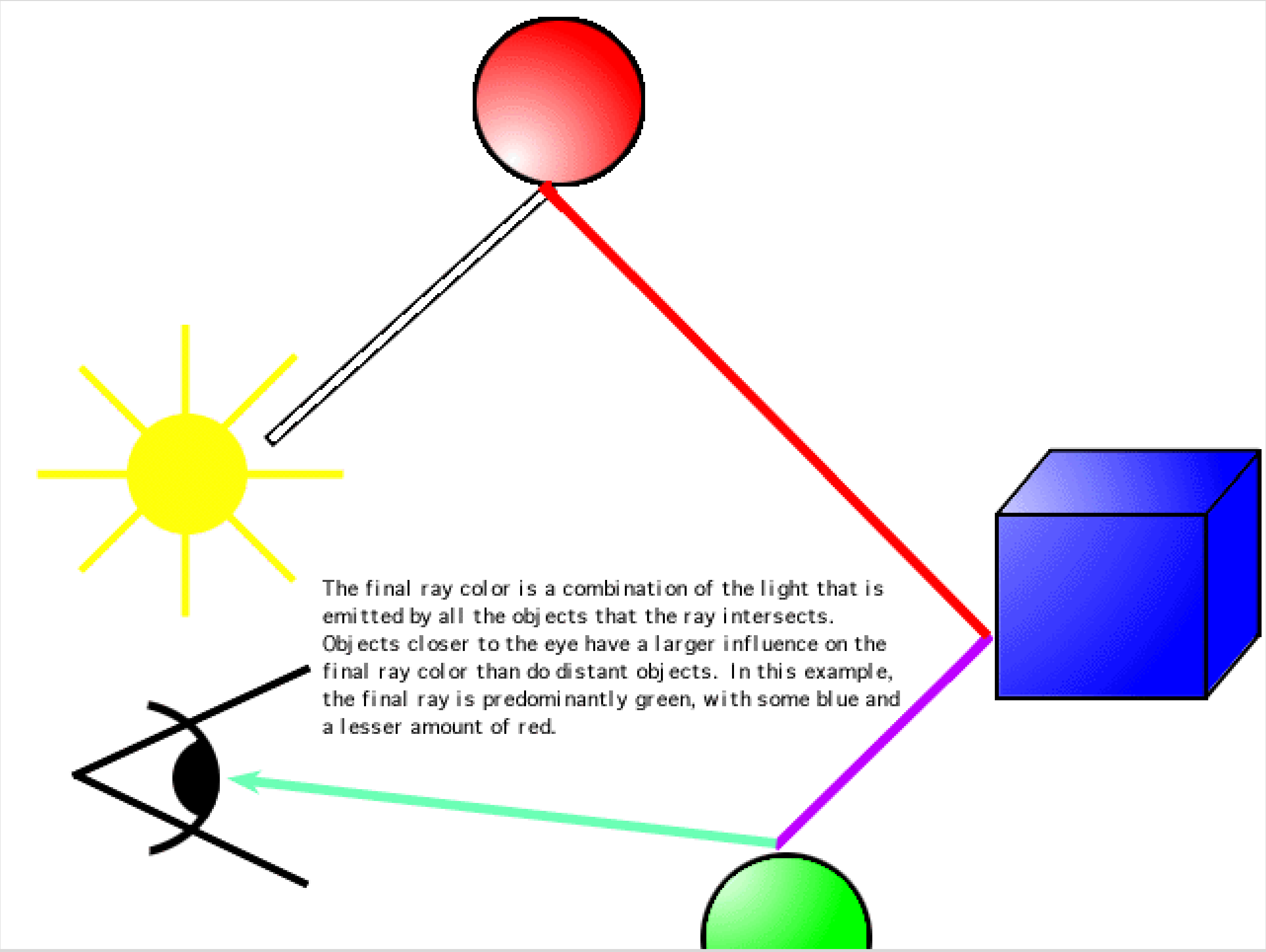


The Ray Tracing Algorithm

```
for each pixel in viewport
{
    determine eye ray for pixel
    intersection = trace(ray, objects)
    colour = shade(ray, intersection)
}
```

```
trace(ray, objects)
{
   for each object in scene
       intersect(ray, object)
   sort intersections
   return closest intersection
}
```




The final ray color is a combination of the light that is emitted by all the objects that the ray intersects. Objects closer to the eye have a larger influence on the final ray color than do distant objects. In this example, the final ray is predominantly green, with some blue and a lesser amount of red.

Ray Tracing Algorithm

```
colour shade(ray, intersection)
{
    if no intersection
        return background colour

    for each light source
        if(visible)
            colour += Phong contribution

    if(recursion level < maxlevel and surface not diffuse)
    {
        ray = reflected ray
        intersection = trace(ray, objects)
        colour += ρrefl*shade(ray, intersection)
      }
    return colour
}
```

Example RayTracing Question

a) Provide an outline of the basic ray-tracing algorithm (You are encouraged to use pseudocode and drawings to illustrate various steps). **[11 marks]**

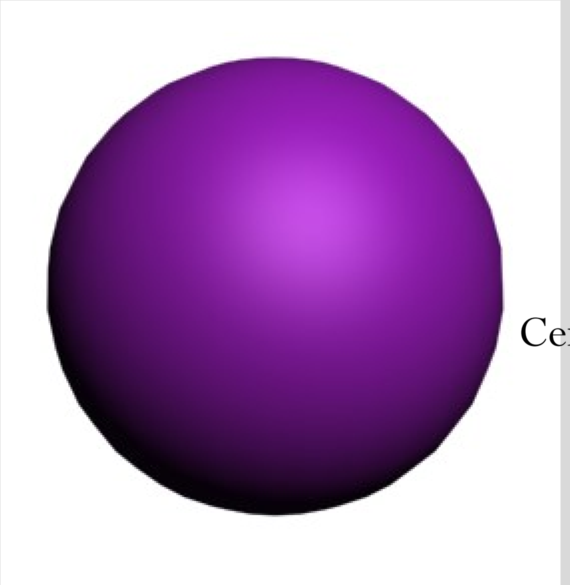
b) Mention a reason for why ray tracing is computationally expensive and a technique that can be used to reduce this cost. **[2 marks]**

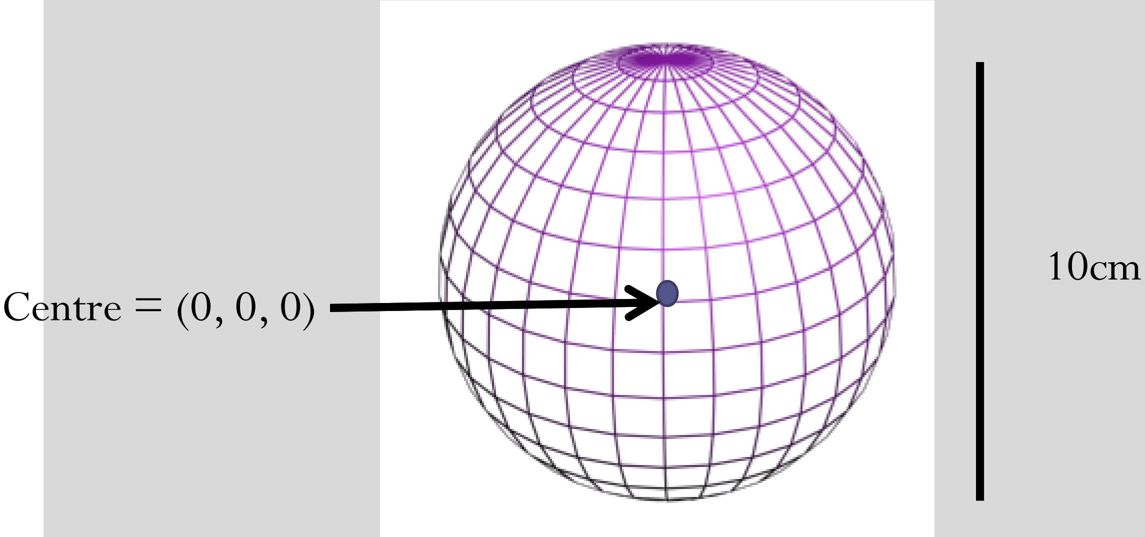
c) There are several different factors that determine the colour of an object at a specific point. Compare local and global illumination of that point, and view dependent and view independent solutions to determining it. **[4 marks]**

d) Outline an algorithm to find the intersection between a ray and a sphere. (Illustrate your answer with an example ray and sphere). **[8 marks]**

The Sphere

- A sphere of center (Cx, Cy, Cz) with radius r is given by: $f(x,y,z) = (x - C_x)^2 + (y - C_y)^2 + (z - C_z)^2 - r^2 = 0$
- Question: is the point (5,1,0) on this sphere?





The Sphere

- A sphere object is defined by its center C and its radius *r*.
- *Implicit Form:* $f(\vec{v}) = |\vec{v} - C|^2 - r^2 = 0$

  $$f(x, y, z) = (x - C_x)^2 + (y - C_y)^2 + (z - C_z)^2 - r^2 = 0$$

- *Explicit Form:* $x = f_x(\theta, \phi) = C_x + r\sin\theta\cos\phi$

  $$y = f_y(\theta, \phi) = C_y + r\cos\theta$$

  $$z = f_z(\theta, \phi) = C_z + r\sin\theta\sin\phi$$

- We can use either form to determine the intersection; we will choose the implicit form.

Ray Sphere Intersection

- All points on the ray are of the form: $\mathbf{r}ay = O + t\vec{d} \quad t \geq 0$
- All points on the sphere satisfy:

$$(x - C_x)^2 + (y - C_y)^2 + (z - C_z)^2 - r^2 = 0$$

- any intersection points (= points shared by both) must satisfy both, so substitute the ray equation into the sphere equation and solve for *t*:

$$([O_x + td_x] - C_x)^2 + ([O_y + td_y] - C_y)^2 + ([O_z + td_z] - C_z)^2 - r^2 = 0$$

ray equation

Problem

$$\left(\left[O_x + t d_x\right] - C_x\right)^2 + \left(\left[O_y + t d_y\right] - C_y\right)^2 + \left(\left[O_z + t d_z\right] - C_z\right)^2 - r^2 = 0$$

ray equation

- Expand the first term
  - remember $(a-b)^2 = a^2 - 2ab + b^2$
- Rearrange into:

$$At^2 + Bt + C = 0$$

Ray Sphere Intersection

- Rearrange and solving for *t* leads to a *quadratic form* (which is to be expected as the sphere is a quadratic surface):

$$At^2 + Bt + C = 0$$
$$A = \left(d_x^2 + d_y^2 + d_z^2\right) = 1$$
$$B = 2d_x\left(O_x - C_x\right) + 2d_y\left(O_y - C_y\right) + 2d_z\left(O_z - C_z\right)$$
$$C = \left(O_x - C_x\right)^2 + \left(O_y - C_y\right)^2 + \left(O_z - C_z\right)^2 - r^2$$

- We employ the classic quadratic formula to determine the 2 possible values of *t*:

$$t = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A} = \frac{-B \pm \sqrt{B^2 - 4C}}{2}$$

Intersection Classification

- Depending on the number of *real roots* we have a number of outcomes which have nice geometric interpretations:
  - we use the *discrimina*

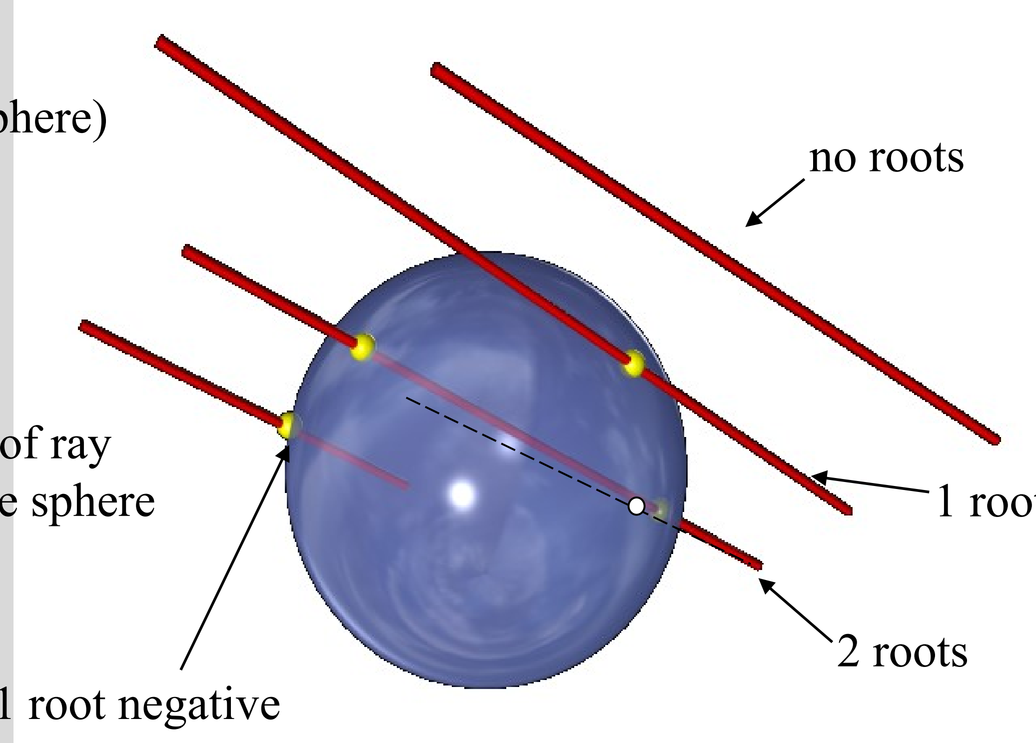
$$d = B^2 - 4C$$

$d = 0 \Rightarrow$ 1 root (ray is tangent to sphere)

$d < 0 \Rightarrow$ no real roots (ray misses)

$d > 0 \Rightarrow$ 2 real roots:

Both positive $\Rightarrow$ sphere in front of ray
One negative $\Rightarrow$ ray origin inside sphere



Example Shader Question

- Given this fragment shader, what would you expect an object to look like that used this shader?

Summary

- Study all topics
  - Graphics Programming
  - Shaders, VBOs, Pipeline, etc.
  - Maths – Linear Algebra
  - & Transformations/ Coordinate Systems
  - Viewing
  - Illumination & Shading
  - Raytracing
  - Mapping
  - Animation
- Remember the new topics since 2015