# Introduction to Graphics

## Cambridge CST Part IA — Tripos Revision Guide

Paper 3 · 2020–2025 Past Questions Analysed

### Topics Covered

1. Images, Pixels & Memory Storage | 2. Ray Tracing | 3. Phong Reflection Model
4. Transformations & Scene Graphs | 5. Rasterisation & Z-Buffer | 6. OpenGL & GLSL
7. Textures | 8. Colour Theory & Colour Spaces | 9. Tone Mapping & Display Encoding

### How to Use This Guide

Each topic section ends with Exam Traps and Key Equations boxes. Past question themes (2020–2025) are highlighted throughout. Equations are formatted for quick scanning under exam conditions.

# 1. Images, Pixels & Memory Storage

## 1.1 What is a Digital Image?

A digital image has two views:

- **Computational (discrete):** a 2D array of pixels, each storing colour values (R, G, B).
- **Mathematical (continuous):** a 2D function I(x, y) giving intensity at every coordinate.

A pixel is a point sample — it has no area, no dimension. It is NOT a box, disk, or light. Pixels are obtained by sampling a continuous function on a rectangular grid.

## 1.2 Pixel Formats & Bit Depth

| | |
|---|---|
| **Grayscale** | 1 channel, 8 bits (256 levels) |
| **Highcolor** | 2 bytes — $2^{16}$ = 65,536 colours |
| **Truecolor** | 3 bytes (24-bit) — 16.8 million colours |
| **Deepcolor** | ≥4 bytes per pixel (used in HDR) |

Why 8 bits? Gamma encoding makes the scale perceptually uniform — fewer bits are needed than in linear space. Without gamma, at least 12 bits per channel would be needed.

Colour banding occurs when bit depth is too low. Dithering (adding noise) can reduce visible banding caused by the Mach band / Chevreul illusion.

## 1.3 Image Memory Layout: Strides

The general pixel index formula is:

```
i(x, y, c) = i_first + x·sx + y·sy + c·sc
```

where sx, sy, sc are strides in the x, y, and colour channel dimensions.

| | |
|---|---|
| **Row-major, grayscale** | i(x, y) = x + y·n  (sx=1, sy=n) |
| **Column-major, grayscale** | i(x, y) = x·m + y  (sx=m, sy=1) |
| **Row-major, interleaved colour** | i(x,y,c) = x·3 + y·3n + c  (sx=3, sy=3n, sc=1) |
| **Row-major, planar colour** | i(x,y,c) = x + y·n + c·n·m  (sx=1, sy=n, sc=n·m) |

For a padded image or ROI: i_first = ox·sx + oy·sy (where ox, oy are the ROI offsets), and strides come from the parent image.

EXAM TIP (2023 Q4): The ExImage class question tested all of this. Know how to set sx, sy, sc for all 4 combinations of row/column-major × interleaved/planar, and how to create an ROI without copying data (just set i_first and borrow the parent's data array).

# 2. Ray Tracing

## 2.1 The Algorithm

Ray tracing simulates light by firing rays from the camera through each pixel:

1. For each pixel, compute a ray from the eye through the pixel centre.
2. Test the ray against every object; find the closest intersection.
3. Shade that intersection point using the Phong model.
4. Optionally spawn reflection/refraction/shadow rays recursively.

> Pinhole camera model: the camera origin corresponds to the pinhole aperture. Rays diverge from this point through a virtual image plane. All points at the same direction from the origin are imaged to the same pixel (no depth of field).

## 2.2 Ray-Object Intersections

### Plane

Ray: $P = O + sD$ ($s \geq 0$). Plane: $P \cdot N + d = 0$.

$$s = -(d + N \cdot O) \;/\; (N \cdot D)$$

For a polygon: intersect with the plane, then check if the point is inside the polygon (2D geometry problem).

### Sphere

Sphere: $(P - C) \cdot (P - C) - r^2 = 0$. Substituting $P = O + sD$:

$$a = D \cdot D, \quad b = 2D \cdot (O - C), \quad c = (O-C) \cdot (O-C) - r^2$$

$$\text{discriminant } d = b^2 - 4ac$$

$$s_1 = (-b + \sqrt{d}) \;/\; 2a, \quad s_2 = (-b - \sqrt{d}) \;/\; 2a$$

$d < 0$: miss. $d = 0$: tangent. $d > 0$: two intersections. Take the smallest positive $s$.

### Cone (2022 Q3)

Implicit equation: $x^2 + z^2 - (r/h)^2 \cdot y^2 = 0$ (apex at origin, base at $y = h$).

Base: intersect ray with plane $y = h$, then check if $x^2 + z^2 \leq r^2$.

Side: substitute ray into implicit equation to get a quadratic in $s$.

$$\text{Normal at } (x, y, z) \text{ on the side: } N = (2x, -2(r/h)^2 y, 2z) \quad \text{[unnormalised]}$$

For transformed cone: transform the ray into object space (multiply by inverse of model transform), intersect there, then transform the normal back using $(M^{-1})^T$.

## 2.3 Ray Tracing Features

| | |
|---|---|
| **Shadows** | Fire a shadow ray from intersection toward each light. If another object is hit, the point is in shadow (no diffuse/specular contribution from that light). |
| **Soft shadows** | Use an area light: distribute shadow rays over the light's area. Average the results — partially blocked points receive partial illumination. |

| | |
|---|---|
| **Reflection** | Spawn a new ray in direction R = D - 2(D·N)N from the intersection point. |
| **Refraction** | Use Snell's law to compute the bent ray direction. A ray can split into reflected + refracted. |
| **Depth of field** | Distribute camera position over a lens disc; only objects at focus distance are sharp. |
| **Motion blur** | Distribute samples over time. |

## 2.4 Anti-Aliasing & Supersampling

Single-ray per pixel → aliasing (jagged edges, missed thin objects). Solutions:

- **Regular grid:** divide pixel into N×N sub-pixels, shoot one ray per sub-pixel. Can still alias.
- **Random:** N rays at random positions. Trades aliasing for noise (less visible).
- **Jittered / stratified:** divide pixel into N sub-regions, one random ray per region. Better than pure random.
- **Poisson disc:** N rays with minimum separation. Best quality, hardest to implement.
- **Adaptive:** shoot a few rays, measure variance; add more only where variance is high.

## 2.5 Time Complexity (2025 Q3)

Without recursion, for P pixels, O objects, L lights:

```
T = O(P · O · L)
```

Outer loop: P pixels. Inner loop: O objects (ray-object tests). Shading: L lights per intersection. Each contributes constant work.

For full spectrum (N spectral bands instead of 3 RGB): multiply by N/3, so O(P · O · L · N).

EXAM TIP (2025 Q3): The question asks about 'loops and their structure'. Be explicit: (1) pixel loop, (2) object loop for intersection, (3) light loop for shading. State complexity for each loop.

# 3. Phong Reflection Model

## 3.1 The Equation

$$I = I_a \cdot k_a + \Sigma_i \, [I_i \cdot kd \cdot \max(0, L_i \cdot N) + I_i \cdot ks \cdot \max(0, R_i \cdot V)^n]$$

| | |
|---|---|
| $I_a \cdot k_a$ | Ambient term. $I_a$ = ambient light intensity, $k_a$ = ambient reflectance. Constant, does not depend on lights or view. Models all indirect inter-surface light. |
| $I_i \cdot kd \cdot (L_i \cdot N)$ | Diffuse (Lambertian). Intensity ∝ cosine of angle between light direction L and surface normal N. View-independent — same from all angles. |
| $I_i \cdot ks \cdot (R_i \cdot V)^n$ | Specular (Phong). R = reflection of L about N. n = roughness coefficient: high n → tight, shiny highlight; low n → broad, dull highlight. View-dependent. |

## 3.2 Key Vectors

- **L:** unit vector from surface point toward light source.
- **N:** unit normal to the surface.
- **R:** perfect reflection direction: R = 2(L·N)N - L (or L mirrored about N).
- **V:** unit vector from surface point toward the viewer/camera.

L·N gives cosθ (angle of incidence). This physically models Lambert's cosine law — a surface receives less irradiance at glancing angles.

## 3.3 When Components Are Zero (2025 Q3)

**Diffuse kd·(L·N) = 0 when:**

- L·N ≤ 0: light is behind the surface (or tangential). Always clamp to max(0, L·N).
- kd = 0: surface has no diffuse reflectance.

**Specular ks·(R·V)ⁿ = 0 when:**

- R·V ≤ 0: viewer is on the wrong side of the reflection direction. Clamp to max(0, R·V).
- L·N ≤ 0: if the light doesn't illuminate the surface at all, specular should also be zero.
- ks = 0: surface has no specular reflectance.

## 3.4 Microfacet Interpretation

Surfaces are modelled as microscopic facets:

- **Diffuse (Lambertian):** facets randomly oriented in all directions — light scattered equally in all directions.
- **Imperfect specular:** facets mostly aligned with surface normal but with some angular spread — a broad highlight.
- **Perfect specular (mirror):** all facets perfectly aligned — one sharp reflection direction.

## 3.5 Shadows in Rasterisation: Shadow Mapping (2023 Q3)

Shadow maps allow shadows in a rasterisation pipeline without ray tracing:

5. Render the depth map from the light's point of view and store as a texture (shadow map).

6. When rendering from camera: transform fragment position p into light-space. Compute (u,v) in shadow map using the light's view-projection matrix.
7. Fragment is in shadow if its depth (in light space) > value stored in shadow map at (u,v).

Finding (u,v): apply light's view and projection matrices V_light, P_light to the world-space fragment position p:

```
p_clip = P_light · V_light · p
(u, v) = (p_clip.x / p_clip.w, p_clip.y / p_clip.w)  — then map from [-1,1] to [0,1]
```

Shadow map artefacts:

- **Self-shadowing / acne:** numerical precision causes a surface to shadow itself. Fix: add a small bias to depth comparison.
- **Aliasing / jagged shadows:** shadow map resolution is too low. Fix: use higher-resolution shadow map or PCF (percentage closer filtering).
- **Peter Panning:** bias is too large — object appears to float. Reduce bias.

## 3.6 Which Phong Terms Can Be Cached? (2023 Q3)

| | |
|---|---|
| **Ambient ($I_a \cdot k_a$)** | Always reusable — constant, independent of camera, lights, and objects. |
| **Diffuse ($I_i \cdot k_d \cdot (L \cdot N)$)** | Reusable if objects and lights don't move. Changes if object moves, light moves. |
| **Specular ($I_i \cdot k_s \cdot (R \cdot V)^n$)** | NOT reusable if camera moves — V depends on viewer direction. Also depends on light and object position. |

EXAM TIP: Specular is view-dependent so it must be recomputed whenever the camera moves. Diffuse is view-independent but still needs recomputation when lights or objects move.

# 4. Geometric Transformations

## 4.1 2D Homogeneous Coordinates

(x, y) in 2D becomes (x, y, w) in homogeneous form. The actual 2D point is (x/w, y/w). Convention: w = 1 for a regular point. w = 0 represents a point at infinity.

## 4.2 Core 3×3 Matrices (2D)

| | |
|---|---|
| **Translation by (tx, ty)** | [1 0 tx / 0 1 ty / 0 0 1] |
| **Scale by (sx, sy)** | [sx 0 0 / 0 sy 0 / 0 0 1] |
| **Rotation by θ** | [cosθ -sinθ 0 / sinθ cosθ 0 / 0 0 1] |
| **Shear (x-axis) by a** | [1 a 0 / 0 1 0 / 0 0 1] |

## 4.3 3D Homogeneous Coordinates

(x, y, z) → (x, y, z, 1). Matrices are 4×4.

| | |
|---|---|
| **Translation (tx,ty,tz)** | [1 0 0 tx / 0 1 0 ty / 0 0 1 tz / 0 0 0 1] |
| **Scale (mx,my,mz)** | [mx 0 0 0 / 0 my 0 0 / 0 0 mz 0 / 0 0 0 1] |
| **Rx (about x, angle α)** | [1 0 0 0 / 0 cosα -sinα 0 / 0 sinα cosα 0 / 0 0 0 1] |
| **Ry (about y, angle β)** | [cosβ 0 sinβ 0 / 0 1 0 0 / -sinβ 0 cosβ 0 / 0 0 0 1] |
| **Rz (about z, angle γ)** | [cosγ -sinγ 0 0 / sinγ cosγ 0 0 / 0 0 1 0 / 0 0 0 1] |

## 4.4 Important Rules

- **Non-commutativity:** order matters! Rotate-then-scale ≠ scale-then-rotate.
- **Concatenation:** multiply matrices together to combine transforms. Apply right-to-left: if you write M = T·R·S, then S is applied first.
- **Standard order:** Scale → Rotate → Translate (SRT). This is the typical convention for placing objects.
- **Arbitrary pivot:** (1) Translate pivot to origin, (2) transform, (3) translate back.

## 4.5 Normal Vector Transformation

Normals do NOT transform with the same matrix as positions (non-orthogonal transforms distort angles).

```
Normal transform G = (M⁻¹)ᵀ  (inverse transpose of the model matrix)
```

For pure rotations: rotation matrices are orthogonal ($R^T = R^{-1}$), so normals can use the same matrix. For scales and shears, use the inverse transpose.

If model matrix is M and we want normal G: $G = (M^{-1})^T$

> Simplification (2025 Q4ii): If M = T·R·S, then $G = (M^{-1})^T$. Translation doesn't affect direction vectors. If scaling is uniform (mx = my = mz), the scale factor cancels and G = R (the rotation alone).

## 4.6 Model Transformation: Placing a Cylinder (2025 Q4i)

Given a unit cylinder at the origin (axis along z, height 2, radius 1), place it with radius r, endpoints A and B:

8. Scale: sx = sy = r (radius), sz = |AB|/2 (half-length).
9. Rotate: align z-axis with direction (B-A)/|B-A|. Use cross products to find rotation.
10. Translate: move to midpoint (A+B)/2.

## 4.7 View Matrix (Look-At)

Camera at c, looking at l, with up vector u:

```
v = (l - c) / |l - c|          (forward direction, left-handed)
r = (v × u) / |v × u|          (right direction)
u' = r × v                     (corrected up direction)
V = [r_x  r_y  r_z  -c·r ]
    [u'_x u'_y u'_z -c·u']
    [v_x  v_y  v_z  -c·v ]
    [0    0    0    1    ]
```

Note: OpenGL is right-handed, so v points away from the scene (-z direction). Signs and cross product order must be adjusted accordingly.

## 4.8 MVP Pipeline

| | |
|---|---|
| **Model (M)** | Transforms object coords → world coords. Different for each object. |
| **View (V)** | Transforms world coords → camera coords (camera at origin, looking at -z). |
| **Projection (P)** | Transforms camera coords → NDC (x, y in [-1, 1]). |

$$\text{Screen coords} = P \cdot V \cdot M \cdot [x\ y\ z\ 1]^T$$

## 4.9 Scene Graphs (2021 Q4)

A scene graph is a tree structure where:

- **Each node** has a local transform and references a primitive (or is an internal node).
- **Child nodes** inherit parent transforms — so moving a parent moves all children.
- **Traversal:** multiply parent transform with node's local transform at each node. draw(parent_T × node.T × node.E).

Example — SARS-CoV-2 model: main sphere → spike cylinders → spike spheres. Rotating the main sphere rotates all spikes.

EXAM TIP (2021 Q4): Draw the scene graph hierarchy, then provide transformation matrices for each node. The spike positions use spherical coordinates — rotate by φ about z, then by θ about y (or vice versa), then translate outward.

# 5. Rasterisation & Z-Buffer

## 5.1 Rasterisation Overview

Rasterisation converts 3D geometry to pixels on screen. It is much faster than ray tracing and is used for real-time rendering.

Algorithm (simplified):

11. Set MVP transformation matrices.
12. For each triangle: transform vertices using MVP. Clip to view frustum.
13. For each fragment (candidate pixel) in the triangle: interpolate attributes; compute colour.
14. If fragment's depth < current depth buffer value, update colour and depth buffers.

## 5.2 Barycentric Coordinates

Given triangle vertices $P_0$, $P_1$, $P_2$, any point P inside can be expressed as:

$$P = \alpha \cdot P_0 + \beta \cdot P_1 + \gamma \cdot P_2, \quad \text{where } \alpha + \beta + \gamma = 1, \quad \alpha, \beta, \gamma \geq 0$$

Compute α and β using the implicit line equation:

$$f_{12}(x, y) = (y_1 - y_2)x + (x_2 - x_1)y + x_1 y_2 - x_2 y_1$$

$$\alpha = f_{12}(x, y) \;/\; f_{12}(x_0, y_0)$$

$$\beta = f_{02}(x, y) \;/\; f_{02}(x_1, y_1)$$

$$\gamma = 1 - \alpha - \beta$$

Point is inside triangle iff $\alpha > 0$, $\beta > 0$, $\gamma > 0$.

Any vertex attribute (colour, normal, texture UV, depth) can be interpolated: $a = \alpha \cdot a_0 + \beta \cdot a_1 + \gamma \cdot a_2$.

## 5.3 Z-Buffer Algorithm

Resolves hidden surface removal:

```
Initialise: colour(x,y) = background, depth(x,y) = z_far
For each fragment (x,y) with depth z:
  if z < depth(x,y) and z > z_near:
    depth(x,y) = z
    colour(x,y) = fragment_colour
```

| | |
|---|---|
| **Z-fighting** | Two surfaces at nearly the same depth — flickering. Fix: increase near plane or increase bit depth. |
| **Precision** | Z-buffer is typically 24 or 32 bits. The usable range is logarithmic — more precision near the camera. |
| **Z-buffer vs. ray tracing** | Z-buffer = rasterisation, O(triangles). Ray tracing naturally handles occlusion via closest-intersection test. |

EXAM TIP (2021 Q3): If you cannot use a Z-buffer (memory constraint), you must sort fragments by depth and paint back-to-front (Painter's algorithm). But this fails for cyclic overlaps. Or you track the visible triangle explicitly as you iterate.

## 5.4 Triangle Mesh Indexing (2021 Q3)

A triangle mesh stores V (vertex positions and attributes) and T (triangle index table). Each row of T contains 3 indices into V.

- **Forward-facing:** in OpenGL, a front-facing triangle has vertices ordered counter-clockwise when viewed from the front.

Lookup algorithm: for a point (x,y), iterate all triangles. Compute barycentric coordinates. If $\alpha,\beta,\gamma \geq 0$, the point is inside — interpolate and return attribute.

# 6. OpenGL & GLSL

## 6.1 Rendering Pipeline

| | |
|---|---|
| **Vertex shader** | Processes each vertex: transforms position (MVP), transforms normals, passes attributes to next stage. PROGRAMMABLE. |
| **Tessellation** | Optional: subdivides patches into more triangles. PROGRAMMABLE. |
| **Geometry shader** | Optional: can create new primitives (fur, shadow volumes). PROGRAMMABLE. |
| **Primitive assembly** | Organises vertices into triangles/lines/points. FIXED. |
| **Clipping** | Removes/clips primitives outside the view frustum. FIXED. |
| **Rasterisation** | Generates fragments from triangles; interpolates vertex attributes. FIXED. |
| **Fragment shader** | Computes colour per fragment; can sample textures, perform tone mapping. PROGRAMMABLE. |
| **Screen buffer** | Final pixel output (colour, depth, stencil). FIXED. |

## 6.2 Vertex vs Fragment Shader Variables (2025 Q4)

**Vertex attributes (per-vertex, can differ between vertices):**

- vec3 position — 3D position in object space
- vec3 normal — surface normal in object space
- vec2 tex_uv — texture coordinates
- vec4 colour — per-vertex colour

**Uniforms (constant for the entire draw call):**

- mat4 mvp_matrix — model-view-projection matrix
- vec3 light_position — position of a light source
- sampler2D diffuse_texture — texture sampler
- float time — animation timer

Key difference (2025 Q4iii): Vertex attributes differ per vertex and are interpolated across triangles by the rasteriser before reaching the fragment shader. Uniforms are constant for the entire draw call — the same value for every vertex and fragment.

## 6.3 Barycentric Coordinates in OpenGL (2025 Q4iv)

In the OpenGL pipeline, barycentric coordinates are used by the rasteriser to interpolate vertex attributes (position, normal, UV, colour) across the interior of each triangle. They are computed automatically in the fixed-function rasterisation stage and used to produce interpolated values passed to the fragment shader.

## 6.4 GLSL Quick Reference

### Data Types

| | |
|---|---|
| **float/int/bool/uint** | Scalars |

| | |
|---|---|
| **vec2, vec3, vec4** | Float vectors; use .xyzw or .rgba |
| **mat2, mat3, mat4** | Float matrices |
| **dvec, ivec, uvec, bvec** | Double/int/uint/bool vector variants |
| **sampler2D, sampler3D** | Texture samplers |

### Storage Qualifiers

| | |
|---|---|
| **in** | Input from previous stage (vertex attribute or interpolated varying) |
| **out** | Output to next stage |
| **uniform** | Set from CPU, constant for entire draw call |
| **const** | Compile-time constant |
| **buffer** | Shader Storage Buffer Object — GPU memory, read/write |

### Swizzling

Select and reorder components: vec4 c = vec4(1,0,0,1); vec3 rgb = c.rgb; vec3 bgr = c.bgr; vec3 gray = c.rrr;

## 6.5 Common OpenGL Artefacts (2022 Q4)

| | |
|---|---|
| **Z-fighting** | Two surfaces at nearly identical depths flicker between frames. Cause: insufficient Z-buffer precision. Fix: move near plane further away, or add a small z-offset (polygon offset). |
| **Texture aliasing** | Blocky (nearest neighbour) or blurry (bilinear upsampling) artefacts. Fix: use appropriate filter or mipmaps for downsampling. |
| **Tearing** | Top and bottom of frame show different animation frames. Fix: enable V-Sync (glfw SwapInterval). |
| **Colour banding** | Visible steps in gradients. Fix: increase bit depth or add dithering. |

## 6.6 Checkerboard Shader (2022 Q4)

Given tex_uv coordinates, create a checkerboard in GLSL:

```
float scale = 10.0;
vec2 tiled = floor(tex_uv * scale);
float checker = mod(tiled.x + tiled.y, 2.0);
vec3 C_d = (checker < 1.0) ? vec3(1,1,0) : vec3(0,0,0);  // yellow
or black
```

# 7. Textures

## 7.1 Texture Mapping

Texture mapping applies an image (texture) to a 3D surface:

15. Define T(u,v): a 2D function from texture coordinates to colour.
16. Assign (u,v) texture coordinates to each vertex of the 3D mesh.
17. During rendering: interpolate (u,v) across each triangle, sample T(u,v), use as colour.

## 7.2 Texture Filtering

| | |
|---|---|
| **Nearest neighbour (upsampling)** | Pick the closest texel. Fast, but blocky artefacts when magnified. |
| **Bilinear (upsampling)** | Interpolate between 4 surrounding texels. Smooth, but slightly blurry. |
| **Mipmapping (downsampling)** | Pre-computed pyramid of lower-resolution versions. Use when pixel covers large area of texture. OpenGL: glGenerateMipmap(). |
| **Trilinear** | Bilinear between two mipmap levels. Smooth transitions. |

## 7.3 Special Texture Types

| | |
|---|---|
| **Bump / Normal map** | Texture stores surface normals instead of colour. Fragment shader uses these to perturb shading — surface appears bumpy without changing geometry. |
| **Displacement map** | Actually displaces vertex positions (needs geometry shader or tessellation). True geometric detail, not just lighting fakery. |
| **Environment map / Cubemap** | Six faces capture the environment in all directions. Used for reflective materials — reflect direction from surface determines which texel to sample. |

# 8. Human Colour Vision & Colour Spaces

## 8.1 Human Visual System

- **Cones (3 types):** responsible for colour vision in daylight. S (short/blue), M (medium/green), L (long/red). Three types → trichromacy.
- **Rods:** responsible for night vision (scotopic). No colour information.
- **Fovea:** high-density cone region at centre of retina. Highest spatial resolution.

Cone response is an integral over all wavelengths:

$$R_S = \int S(\lambda) \cdot L(\lambda)\, d\lambda \quad \text{(where } S(\lambda) = \text{S-cone sensitivity, } L(\lambda) = \text{light spectrum)}$$

Similarly for R_M and R_L. Only these 3 numbers determine perceived colour.

## 8.2 Metamers

Two physically different light spectra that produce identical cone responses are metamers — they look the same.

Test for metamers: compute [L, M, S] cone responses for both spectra. If identical → metamers.

$c_1$ and $c_2$ are metamers iff:

$$\int l(\lambda) c_1(\lambda) d\lambda = \int l(\lambda) c_2(\lambda) d\lambda$$

$$\int m(\lambda) c_1(\lambda) d\lambda = \int m(\lambda) c_2(\lambda) d\lambda$$

$$\int s(\lambda) c_1(\lambda) d\lambda = \int s(\lambda) c_2(\lambda) d\lambda$$

This is why displays work: display primaries are physically different spectra from real-world objects, but they create metameric matches by producing the same cone responses.

## 8.3 CIE XYZ Colour Space

The CIE 1931 XYZ space is the foundation of all colour standards:

- **All-positive colour matching functions** (unlike real cone responses which can be negative).
- **Y** ≈ luminance (light intensity weighted by visual sensitivity).
- **Device-independent** — all other colour spaces are defined relative to XYZ.
- **Linear:** transformations between linear colour spaces use matrix multiplication.

CIE chromaticity diagram: plot x = X/(X+Y+Z), y = Y/(X+Y+Z). Ignores luminance. Pure spectral colours lie on the horseshoe boundary. All real colours are inside.

## 8.4 RGB Colour Spaces

| | |
|---|---|
| **ITU-R BT.709 (sRGB)** | Standard for HD TV and most computer monitors. Primaries chosen for reproducibility with CRT phosphors (now historical). Smaller gamut. |
| **ITU-R BT.2020** | Wide colour gamut standard for 4K/HDR. Primaries near the spectral locus — few real displays can reproduce it fully. Future-proofed. |
| **DCI-P3** | Cinema standard, wider than 709, narrower than 2020. |

| | |
|---|---|
| **Adobe RGB** | Wider than sRGB, popular in photography. |

## 8.5 Converting Between Colour Spaces

All conversions go via CIE XYZ as the device-independent intermediate:

$$\text{RGB}_{709} \rightarrow \text{XYZ}: \quad [X\ Y\ Z]^T = M_{709\rightarrow XYZ} \cdot [R\ G\ B]^T$$

$$M_{709\rightarrow XYZ} = \begin{bmatrix} 0.4124 & 0.3576 & 0.1805 \\ 0.2126 & 0.7152 & 0.0722 \\ 0.0193 & 0.1192 & 0.9505 \end{bmatrix}$$

$$\text{XYZ} \rightarrow \text{RGB}_{709}: \quad M = M_{709\rightarrow XYZ}^{-1}$$

$$\text{RGB}_{709} \rightarrow \text{RGB}_{2020} = M_{2020\rightarrow XYZ}^{-1} \cdot M_{709\rightarrow XYZ} \cdot \text{RGB}_{709}$$

## 8.6 Mapping to a Display with Custom Primaries (2022 Q4, 2024 Q3)

Given measured primary spectra P (N×3 matrix), XYZ sensitivities S_XYZ (N×3):

18. Find XYZ of each primary: M_prim = S_XYZ^T · P (3×3 matrix mapping primary RGB → XYZ)
19. Full conversion: native_RGB = $M_{prim}^{-1} \cdot M_{709\rightarrow XYZ} \cdot \text{input\_RGB}_{709}$

## 8.7 Perceptually Uniform Colour Spaces

| | |
|---|---|
| **CIE L*a*b* (Lab)** | Approximately uniform — equal Euclidean distances correspond to equal perceived colour differences. L* = lightness, a* = red-green, b* = blue-yellow. Good for error metrics. |
| **CIE L*u*v* (Luv)** | Similar to Lab. u'v' chromaticity is more perceptually uniform than xy. |
| **HSV / HLS** | Perceptually intuitive for artists (hue, saturation, value/lightness). Not uniform. Easy colour pickers. |

EXAM TIP (2024 Q3a): HSV/HLS for colour pickers; CIE Lab for error metrics (perceptually uniform); fp16 or log-encoded for HDR textures.

# 9. Display Encoding & Tone Mapping

## 9.1 Scene-Referred vs Display-Referred

| | |
|---|---|
| **Scene-referred** | Linear floating-point values representing physical light intensities. Very wide dynamic range (e.g. 0.005 to 100,000 cd/m²). Used in rendering, HDR cameras. |
| **Display-referred** | Encoded integers (e.g. 0–255) representing values a specific display can show. Narrow dynamic range. Used in JPEG, PNG, video. |

## 9.2 Gamma Correction / Display Encoding

Displays use gamma encoding because:

- **Perceptual uniformity:** 8-bit gamma-encoded values distribute perceptually equally-spaced brightness levels. Linear encoding would need 12+ bits for the same quality.
- **Historical:** also matched CRT electron gun response curve.

OETF (Opto-Electrical Transfer Function) — camera/rendering to display signal:

```
V' = V^(1/γ)      (encoding, γ ≈ 2.2 for sRGB)
```

EOTF (Electro-Optical Transfer Function) — display signal to light output:

```
V = V'^γ          (display decoding)
```

Full pipeline: linear rendering → gamma encode (OETF) → store/transmit → display decodes (EOTF) → light output.

## 9.3 Linear-to-Display Encoding Equation (2020 Q3e)

Map linear value x (arbitrary range, where x_white maps to peak display value) to encoded pixel [0–255]:

```
pixel = 255 · (x / x_white)^(1/2.2)
```

Breakdown: divide by x_white to normalise to [0,1]; apply gamma (1/2.2) for perceptual encoding; multiply by 255 for integer range.

## 9.4 Standards Summary

| | |
|---|---|
| **SDR (Standard Dynamic Range)** | ITU-R 709 colour space, 2.2 gamma or sRGB OETF, 8–10 bits per channel. |
| **HDR (High Dynamic Range)** | ITU-R 2020 colour space, PQ (Perceptual Quantiser) or HLG (Hybrid Log-Gamma) OETF, 10–12 bits. |
| **sRGB** | ≈ ITU-R 709 with a piece-wise OETF (linear below 0.003, gamma above). The standard for most web content. |

## 9.5 Tone Mapping

Tone mapping compresses wide scene dynamic range into displayable range. Needed because:

- **Dynamic range mismatch:** real scenes span ~$10^{-3}$ to $10^{8}$ cd/m², displays output 0–1000 cd/m².
- **Other reasons:** colour grading, simulating camera/film look, adapting to viewing conditions.

### Sigmoidal Tone Mapping

$$R'(x,y) = R(x,y) \;/\; (L_w/a + R(x,y))$$

Where $L_w$ = geometric mean of luminance (scene key):

$$L_w = \exp(1/N \cdot \Sigma \ln(L(x,y)))$$

Parameter a controls exposure (smaller a → brighter image). This mimics photographic film response.

### Tone Curve Design

- **Best case (slope = 1):** no contrast compression needed — only possible if dynamic range fits the display.
- **Practical case:** S-curve (sigmoidal) — preserve contrast in midtones, compress highlights and shadows.
- **Global vs local:** global applies the same curve to all pixels; local adapts based on surrounding context.

## 9.6 Why Glare Enhances Realism (2020 Q3c, 3d)

Glare (light bloom) occurs in eyes and cameras when bright light scatters in the optical system. Adding simulated glare around very bright sources:

- **Enhances perceived brightness:** viewers interpret glare as evidence that the source must be very bright — beyond display capability.
- **Only applied above display maximum:** values within displayable range should render as-is (no artificial alteration). Glare is simulated only for values that cannot be reproduced directly — it is a psychovisual substitute for that lost information.

# 10. Past Paper Question Analysis

## 2020 Q3: Colour & Display Encoding

| | |
|---|---|
| **(a) Scene → display mapping** | Bridges the gap between scene-referred (physical light, float) and display-referred (integer pixel, limited range). Allows HDR scenes to be shown on SDR displays. |
| **(b) SDR vs HDR encoding** | SDR uses gamma (2.2). HDR uses PQ (Perceptual Quantiser) or HLG — these encode a much larger luminance range efficiently. |
| **(c) Glare enhances realism** | Creates perception of brightness beyond display maximum — psychovisual trick. |
| **(d) Why glare for clipped values only** | Values within display range should render accurately. Glare only compensates for information that cannot be displayed. |
| **(e) Encoding equation** | pixel = 255 · (x / x_white)^(1/2.2) |
| **(f) Testing metamers** | Compute L, M, S cone responses for both spectra. Metamers iff all three integrals match. |

## 2020 Q4: Phong + Reflection Geometry

| | |
|---|---|
| **(a)(i) Diagram** | Draw surface, N, $L_1$, $L_2$, $R_1$, $R_2$, V vectors. |
| **(a)(ii) Diffuse term** | Lambertian reflection — independent of viewer. Controlled by kd (reflectance) and L·N. |
| **(a)(iii) Specular term** | Phong approximation — view-dependent. Controlled by ks and n (roughness). Higher n = tighter highlight. |
| **(a)(iv) Ambient term** | Constant illumination representing indirect light. Controlled by ka. No dependence on direction. |
| **(a)(v) L·N meaning** | Cosθ — Lambert's cosine law. Models reduced irradiance at glancing angles. |
| **(a)(vi) Microfacets** | Diffuse: random facets. Imperfect specular: mostly aligned + spread. Perfect specular: perfectly aligned. |
| **(b) Ray reflection geometry** | Use reflection formula for ray at B. Find intersection with plane of C: solve parametric ray equation with plane equation. |

## 2021 Q3: Triangle Mesh & Z-Buffer

| | |
|---|---|
| **(a) Index table** | List triangles with CCW winding for front-facing. |
| **(b) lookup_a pseudocode** | Iterate triangles → compute barycentric coords → check if inside → interpolate attribute. Return -1 if not found. |
| **(c) With depth & occlusion** | Cannot use Z-buffer. Track minimum z of visible triangle: iterate triangles, check if point is inside, if z < current_min_z, update. Return attribute of winner. |

## 2021 Q4: Scene Graph & Transformations

| | |
|---|---|
| **(a) Scene graph** | Root: main sphere. Children: N spike assemblies. Each spike: cylinder node + sphere node as children. |
| **(b) Transformation matrices** | For each spike at (φ,θ): Rz(φ)·Ry(θ)·T(0,1,0) places cylinder on |

| | |
|---|---|
| | sphere surface. Scale(0.025, 0.025, 0.05) for cylinder dimensions. Further child sphere at tip. |
| **(c) Random spike placement** | Rejection sampling: generate random spherical coords, check all pairwise distances > min_dist. Or use Poisson disc sampling on the sphere surface. |

## 2022 Q3: Cone Ray Intersection

| | |
|---|---|
| **(a)(i) Base intersection** | Intersect ray with plane y=h; check if $x^2+z^2 \leq r^2$. |
| **(a)(ii) Side intersection** | Substitute ray into cone implicit equation; solve quadratic in s. |
| **(b) Normal at surface** | Gradient of implicit function: N = ∇f = (2x, -2($r^2/h^2$)y, 2z) — then normalise. |
| **(c) Transformed cone** | Apply inverse transform to ray: O' = $(T \cdot Rz \cdot Rx)^{-1} \cdot O$, D' = $(T \cdot Rz \cdot Rx)^{-1} \cdot D$. Then intersect with canonical cone. Normal back-transform: $(M^{-1})^T \cdot N$. |

## 2022 Q4: OpenGL Artefacts, GLSL, Colour

| | |
|---|---|
| **(a) Artefacts** | Identify (e.g. Z-fighting, aliasing) from screenshots. State cause and fix. |
| **(b) Checkerboard shader** | floor(tex_uv * scale); checker = mod(sum, 2); choose yellow or black based on value. |
| **(c) Colour conversion** | linear R'G'B' (ITU-R 2020) → XYZ ($M_{2020 \to XYZ}$) → native display XYZ → solve for primaries → gamma-encode. |

## 2023 Q3: Phong + Shadow Mapping

| | |
|---|---|
| **(a)(i) Negative dot products** | Clamp L·N and R·V to max(0, ...). Negative means light/view is behind surface. |
| **(a)(ii) Shadows in ray tracing** | Shadow rays from intersection toward each light source. |
| **(a)(iii) Soft shadows** | Distribute shadow rays over area light; average results. |
| **(a)(iv) Caching Phong terms** | Ambient: always cacheable. Diffuse: cacheable if lights+objects static. Specular: not cacheable if camera moves. |
| **(b) Shadow mapping** | (i) depth > shadow map value → in shadow. (ii) (u,v) from P_light·V_light·p. (iii) Artefacts: self-shadowing/acne, aliasing, peter-panning. |

## 2023 Q4: Image Class Implementation

| | |
|---|---|
| **(a) Constructor strides** | Depends on row/column-major × interleaved/planar. |
| **(b) get_index** | i = first_pixel + x*sx + y*sy + cc*sc |
| **(c) ROI constructor** | Set first_pixel = ox*parent.sx + oy*parent.sy; copy strides from parent; share data array. |
| **(d) Linear vs encoded** | Store linear values for correct arithmetic (filtering, blending). Convert to encoded only for display output. |
| **(e) Colour conversion** | Non-standard RGB → XYZ (via CIE matching fns and primaries |

| | |
|---|---|
| | spectra) → BT.709 RGB (via matrix) → gamma encode. |

## 2024 Q3: Colour Space Selection & Conversion

| | |
|---|---|
| **(a)(i) Colour picker** | HSV or HLS — intuitive hue/saturation/value axes for artists. |
| **(a)(ii) Camera error metric** | CIE Lab — perceptually uniform, so Euclidean distance = perceived difference. |
| **(a)(iii) HDR textures** | Log-encoded (e.g. OpenEXR half-float or RGBE) or PQ — efficient range, preserves HDR values. |
| **(b)(i) BT.709 primaries motivation** | Matched capabilities of CRT phosphors available in 1970s/80s. |
| **(b)(ii) BT.2020 gamut motivation** | Future-proofing for better display technology. Chosen at the spectral locus for maximum possible gamut. |
| **(c) Custom display matrix** | native_RGB = $(S\_XYZ^T \cdot P)^{-1} \cdot M_{709 \to XYZ} \cdot RGB_{709}$ |

## 2024 Q4: AR Headset Rendering

| | |
|---|---|
| **(a)(i) 6-bit colour** | Use dithering to add noise and reduce visible banding artefacts. |
| **(a)(ii) Large pixels (low res)** | Use anti-aliasing (supersampling or TAA). Foveated rendering (higher quality at gaze point). |
| **(a)(iii) See-through display** | Measure ambient luminance; adjust display brightness to maintain visibility. Additive mixing means display 'black' is transparent — compensate with tone mapping. |
| **(b) Stereo view matrices** | Left eye at e_L; right eye at e_R = e_L + $\hat{r} \cdot$ d_IOD where $\hat{r} = (\hat{v} \times \hat{u})/\lvert\hat{v} \times \hat{u}\rvert$. Construct look-at matrix for each. |

## 2025 Q3: Ray Tracing Theory

| | |
|---|---|
| **(a) Pinhole camera** | Camera origin = pinhole. Rays diverge from origin through image plane. No depth of field. |
| **(b) Time complexity** | O(P·O·L): P pixels × O objects (intersection) × L lights (shading). |
| **(c) Shadow rays & colour in shadow** | Shadow ray from hit point to each light. If occluded: that light contributes 0 diffuse + 0 specular. Only ambient remains: I = Ia·ka. |
| **(d) When specular/diffuse = 0** | Diffuse: L·N ≤ 0 (light behind surface), or kd = 0. Specular: R·V ≤ 0 (view wrong side), or L·N ≤ 0, or ks = 0. |
| **(e) Specular vs diffuse** | Specular: view-dependent, sharp highlight, high n → small spot. Diffuse: view-independent, uniform across surface (only depends on angle to light). |
| **(f) Full spectrum rendering** | Shoot separate rays for N spectral bands (or weight by wavelength). Complexity: O(P·O·L·N). N ≈ 400–700 bands for 1nm resolution. |

## 2025 Q4: Transformations & OpenGL Pipeline

| | |
|---|---|
| **(a)(i) Cylinder transform** | Given unit cylinder (radius 1, ends at z=0 and z=2): Scale(2, 2, \|AB\|/2) → Rotate (z-axis to (B-A)/\|B-A\|) → Translate((A+B)/2). |
| **(a)(ii) Normal transform** | G = $(M^{-1})^T$. If M = T·R·S_uniform, simplifies to R (since uniform scale |

| | |
|---|---|
| | cancels and translation doesn't affect normals). |
| **(b)(i) Z-buffer** | Stores depth (z-value after projection) of closest fragment for each pixel. Needed to resolve hidden surfaces. Computed from vertex depth interpolated via barycentric coordinates. |
| **(b)(ii) Vertex attrs & uniforms** | Attributes: position, normal, UV, colour. Uniforms: MVP matrix, light pos, material colour, time. |
| **(b)(iii) Uniform vs attribute** | Attribute: per-vertex, interpolated across triangles. Uniform: same value for entire draw call. |
| **(b)(iv) Barycentric coords** | Express interior point as weighted sum of vertices. Used by rasteriser to interpolate all vertex attributes (normal, UV, colour) before fragment shader. |

# 11. Key Equations Quick Reference

## Phong Model

$$I = Ia \cdot ka + \Sigma_i \, [Ii \cdot kd \cdot \max(0, Li \cdot N) + Ii \cdot ks \cdot \max(0, Ri \cdot V)^n]$$

$$R = 2(L \cdot N)N - L \quad \text{(reflection vector)}$$

## Ray-Object Intersections

$$\text{Ray: } P = O + sD, \quad s \geq 0$$

$$\text{Plane: } s = -(d + N \cdot O) \,/\, (N \cdot D)$$

$$\text{Sphere: } a = D \cdot D, \quad b = 2D \cdot (O - C), \quad c = (O - C) \cdot (O - C) - r^2, \quad s = (-b \pm \sqrt{(b^2 - 4ac)}) \,/\, 2a$$

## Image Memory

$$i(x, y, c) = i\_first + x \cdot sx + y \cdot sy + c \cdot sc$$

## Barycentric Coordinates

$$\alpha = f_{12}(x, y) / f_{12}(x_0, y_0), \quad \beta = f_{02}(x, y) / f_{02}(x_1, y_1), \quad \gamma = 1 - \alpha - \beta$$

$$f\_AB(x, y) = (yA - yB)x + (xB - xA)y + xA \cdot yB - xB \cdot yA$$

## Transformations (3D Homogeneous)

$$\text{Final} = P \cdot V \cdot M \cdot \text{vertex}$$

$$\text{Normal transform: } G = (M^{-1})^T$$

$$\text{View (look-at): } V = [r^T \; -c \cdot r \;/\; u'^T \; -c \cdot u' \;/\; v^T \; -c \cdot v \;/\; 0 \; 0 \; 0 \; 1]$$

## Colour

$$\text{Metamer test: } \int l(\lambda) c_1(\lambda) d\lambda = \int l(\lambda) c_2(\lambda) d\lambda \quad \text{(same for m and s)}$$

$$RGB_{709} \rightarrow XYZ: [0.4124 \; 0.3576 \; 0.1805 \;/\; 0.2126 \; 0.7152 \; 0.0722 \;/\; 0.0193 \; 0.1192 \; 0.9505]$$

$$\text{Luma: } Y' = 0.2126R' + 0.7152G' + 0.0722B' \quad \text{(gamma-corrected values)}$$

## Display Encoding

$$\text{Gamma encode (OETF): } V' = V^{(1/\gamma)}, \quad \text{typically } \gamma = 2.2$$

$$\text{Pixel value: } p = 255 \cdot (x \,/\, x\_white)^{(1/\gamma)}$$

## Tone Mapping (Sigmoidal)

$$R'(x, y) = R(x, y) \,/\, (L\_w/a + R(x, y))$$

$$L\_w = \exp(1/N \cdot \Sigma\_\{x,y\} \ln(L(x, y))) \quad \text{(geometric mean)}$$

## Ray Tracing Complexity

Without spectrum: $O(P \cdot O \cdot L)$

With N spectral bands: $O(P \cdot O \cdot L \cdot N)$

# 12. Exam Tips & Common Pitfalls

### General Advice

- Always max(0, L·N) and max(0, R·V) — never allow negative contribution from diffuse or specular.
- Transformation order is CRITICAL. SRT means S applied first, T last. Write matrix multiplication right-to-left.
- Normals use $(M^{-1})^T$, not M. For uniform scale + rotation only, this simplifies to R.
- For colour conversion questions: always go via XYZ. Never convert directly between RGB spaces without XYZ as intermediate.
- Linear vs display-encoded: store linear for computation, encode only for display/storage. Filtering/blending must happen in linear space.
- In a shadow, only ambient remains: I = Ia·ka.
- Specular is VIEW-DEPENDENT; diffuse is VIEW-INDEPENDENT.
- Z-buffer precision issue: Z-fighting occurs when near and far planes are too far apart. Reduce ratio far/near.
- Scene graph: transforms accumulate by multiplication down the hierarchy. Moving a parent moves all children.
- Metamers: two spectra that produce identical L, M, S cone responses. Test with three integrals.

### Topics That Appear Every Year

- Q3 (rkm38): Colour / tone mapping / display encoding — appeared 2020, 2022, 2023, 2024.
- Q4 (rkm38): Transformations and geometry — appeared every year in some form.
- Phong model — appears in almost every paper, often paired with shadows.
- Coding questions (pseudocode or GLSL) — be comfortable writing clear, correct code.
- Image memory (strides, ROI) — tested 2021 and 2023.