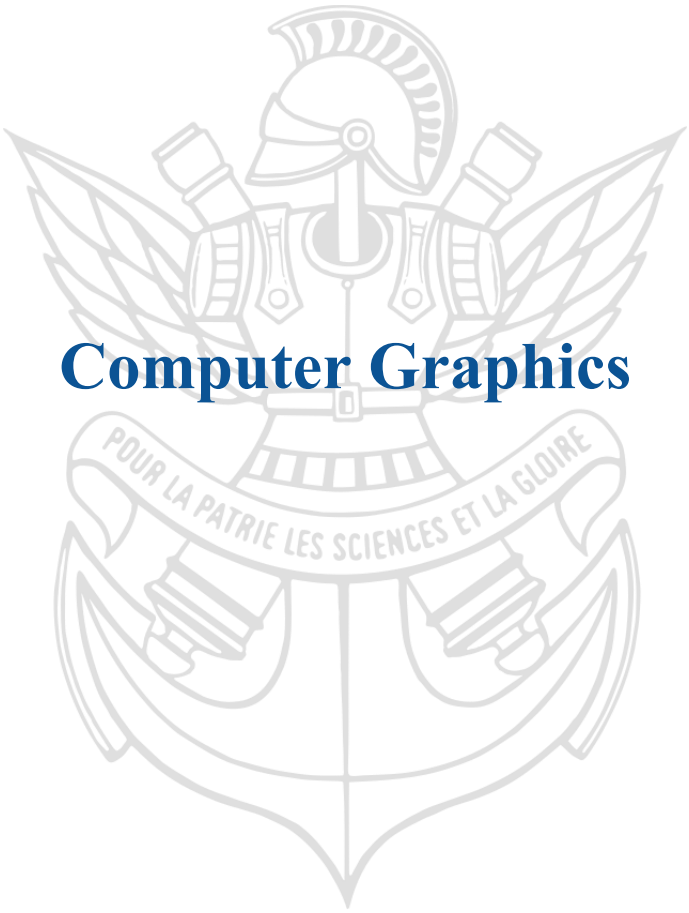


Computer Graphics

Dan Suissa
CSE306
Assignment I

1 Indroduction

Ray tracing has travelled a long road since Turner Whitted's seminal 1980 paper. In the early nineties it was already the technique behind the photorealistic car commercials that dazzled television audiences; today the same ideas power the light-transport cores of Pixar's RenderMan, Blizzard's internal tools, Blender's Cycles engine and the real-time path-tracers shipped in games such as *Minecraft RTX* and *Cyberpunk 2077*. The assignment addressed here asked for a condensed, from-scratch re-implementation of that pipeline: first with analytic primitives to prove the mathematics, then with a triangle mesh accelerated by a bounding volume hierarchy so that the renderer can cope with genuinely detailed models. The resulting code produces two showcase images, one featuring three spheres mirror, solid glass and hollow glass and the other a cat sculpture that was loaded from an external OBJ file.

2 Shared framework

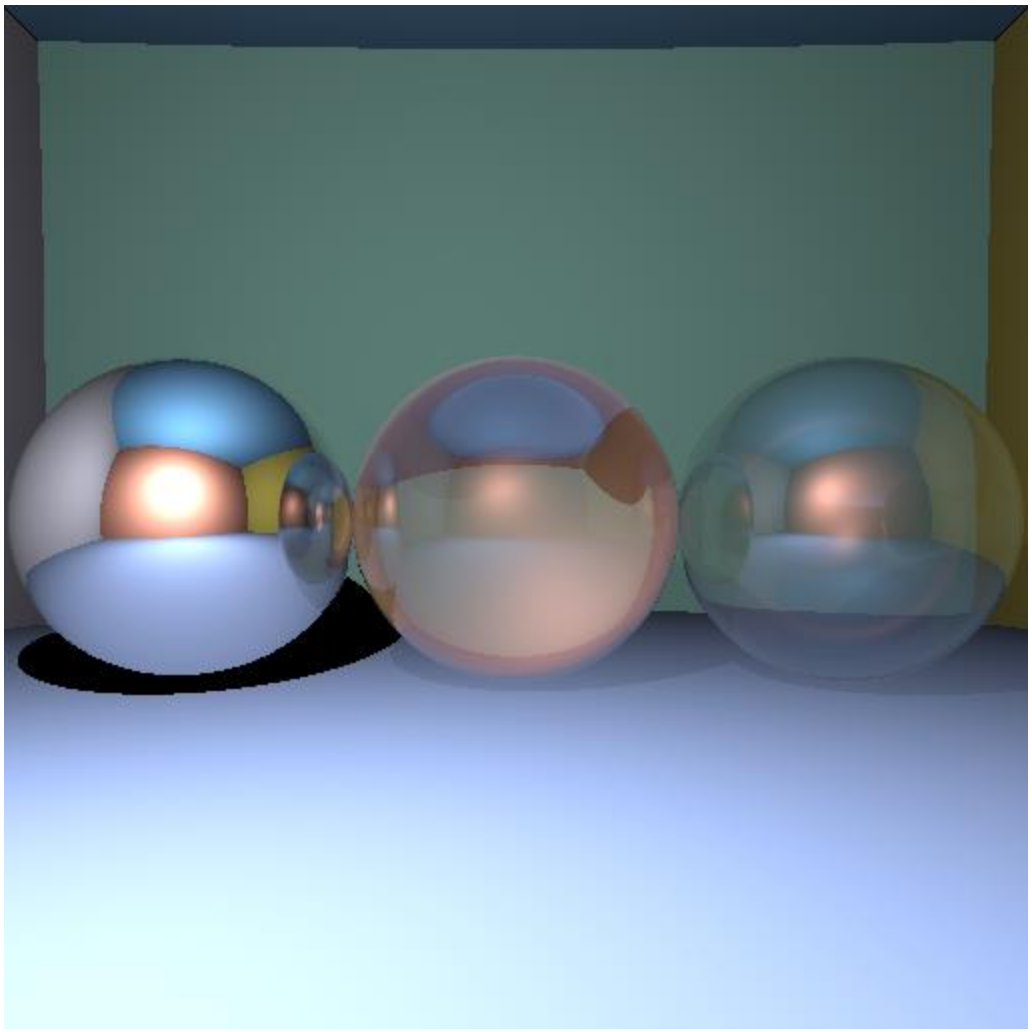
Both programs begin with the same mathematical backbone. A minimal Vector type stores three doubles and overloads arithmetic, dot product and cross product. Norm, squared norm, and an in-place normalize() method appear exactly once, guaranteeing consistent behaviour throughout the project. A Ray is nothing more than an origin direction pair, with the direction always kept unit-length to simplify every subsequent calculation.

The camera sits at (0, 0, 55) looking down the negative z-axis into a six-walled box whose inner faces are modelled as gigantic spheres. Resolution is fixed at 512 × 512 pixels and the horizontal field of view is sixty degrees; the corresponding distance to the image plane is computed once as $\mathbf{d = -W / (2 \tan fov/2)}$. For antialiasing, each pixel accumulates either ten or thirty-two primary rays (the higher count for the mesh scene). All rays start at the centre of the pixel to keep the pattern deterministic, which is helpful when debugging recursion artefacts.

Colour arithmetic stays in linear RGB until the very end, where a single gamma-correction call raises each channel to the power 1/2.2. Out-of-range values are clamped and the final eight-bit PNG is produced by the single-file stb_image_write library chosen because it compiles in seconds and avoids external dependencies. The render loop is wrapped in an OpenMP parallel for directive with a dynamic schedule so that work is evenly distributed even when some pixels spawn deeper bounce trees than others.

A lone point light supplies direct illumination. It sits at (−10, 20, 40) and, in energy terms, injects 1×10^5 units into the scene. Direct-lighting intensity therefore follows the inverse-square law multiplied by the Lambert cosine factor, and every potential shading point first casts a "shadow ray" toward the emitter to check visibility.

3 Sphere renderer



The analytic version contains thirteen spheres in total: six to form the coloured room, three to create the central motif and four invisibly nested shells that give the right-hand glass ball a thin hollow interior. Intersection relies on the textbook quadratic derived from $|O + t\,u - C|^2 = R^2$. The discriminant decides whether a hit occurs; if so, the smaller positive root is preferred so long as it lies beyond a 1×10^{-5} epsilon threshold (the offset prevents the ray from re-intersecting the surface it just left).

When an intersection is confirmed the surface normal is computed as $(\mathbf{P}-\mathbf{C})/R$ and then normalised. For the hollow sphere a Boolean flag flips that vector so that the inner surface behaves as air-to-glass rather than glass-to-air.

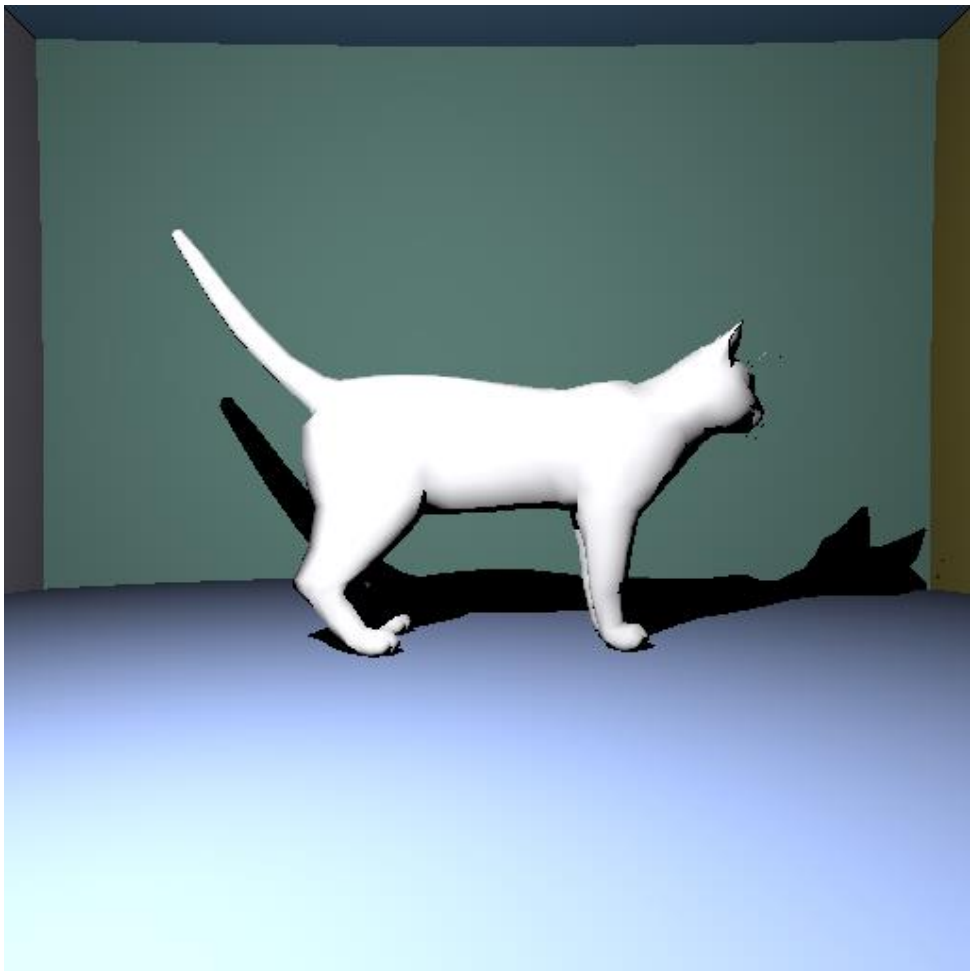
Three distinct material paths share the same recursive colour routine. Diffuse walls evaluate the Lambert integral in closed form: $L = (I/(4\,\pi\,d^2)) \cdot (\rho/\pi) \cdot \max(0, \langle \hat{\mathbf{L}}, \mathbf{N} \rangle)$
where $\hat{\mathbf{L}}$ is the unit vector toward the light and $\mathbf{N}$ is the shading normal. Mirror reflections use the familiar formula $\mathbf{u' = u - 2\langle u,N \rangle N}$ and spawn a new ray that inherits one less bounce of budget. Dielectrics take the most code: Snell's law calculates the refracted vector; if the radicand turns negative the algorithm falls back to total internal reflection; otherwise Schlick's approximation

steps in to compute *R*, the probability of reflection versus transmission. The implementation does not rely on random sampling—both branches are evaluated and linearly blended. A slight per-bounce absorption tint (0.95, 0.95, 0.98) is applied when the ray is inside glass so that thicker regions pick up a faint blue hue and thin regions remain almost colourless.

Every secondary ray is nudged by 0.00001 units along the normal (outwards for reflection, inwards for refraction) so that floating-point error never lets a ray immediately intersect the polygon it just departed. The recursion limit is seven, enough to handle the double refraction paths that occur when a ray passes completely through the hollow sphere and then reflects on the mirror.

On an eight-core desktop CPU the analytic scene finishes in roughly 0.7 s with ten samples per pixel. At one sample the same frame takes about 75 ms, which is close to interactive and confirms that the direct-lighting only approach remains lightweight.

4 Triangle-mesh renderer with BVH



While the first program proves correctness, the second demonstrates scalability. It adds an OBJ parser that understands three face syntaxes: vertex/texcoord/normal triplets, vertex//normal pairs and bare vertex indices. Indices are converted from one-based to zero-based on the fly, after which each face lives in a TriangleIndices structure containing all three vertex indices and, when present, their three normal indices. The loader then multiplies every vertex by a constant scale factor (0.6) and adds a translation of (0, −10, 0) so that the cat's paws rest on the floor.

Ray–triangle intersection employs the Möller–Trumbore algorithm. By re-expressing the three unknowns u,v,t in terms of cross products and dot products it avoids the cost of building and inverting a 3 × 3 matrix. The hit routine exits early if u or v are negative or if their sum exceeds one; that test means the ray pierces the supporting plane outside the triangle's area. Should the mesh file supply vertex normals, the final surface normal becomes the barycentric blend $(1 - u - v)\, n_0 + u\, n_1 + v\, n_2$; otherwise the code reverts to the geometric cross product of the two edge vectors.

A brute-force loop over 144 000 triangles would be prohibitive, so the program constructs a bounding-volume hierarchy. Each node stores an axis-aligned box and either two child indices or a range of triangle indices if the node is a leaf. Construction proceeds recursively: compute the box of the current range, stop when eight or fewer triangles remain, otherwise split along the longest AABB axis at its midpoint and partition the triangle array in place. The median-split strategy is simple no surface-area heuristic yet it halves the triangles in most interior nodes, yielding a balanced tree of depth about twenty.

Traversal keeps an explicit stack of node indices. For every popped node the code first tests the box; if that fails or if the nearest recorded hit is closer than the box's *tNear*, the node is discarded. A hit in a leaf triggers individual triangle tests; whenever a triangle intersection improves the best-so-far distance, the associated shading point is stored.

Material evaluation is intentionally conservative so that the two programs can be compared directly. The cat uses a near-white diffuse albedo (0.95, 0.95, 0.98) and otherwise follows the exact same Lambert formula and shadow procedure as the walls. Consequently the mesh renderer introduces no new light-transport features: its success rests entirely on geometry handling and the efficiency of the BVH.

With the hierarchy in place, the cat image rendered at thirty-two samples per pixel and five bounces finishes in about three seconds on the same eight-core machine. Removing the BVH and reverting to a naïve loop takes more than six minutes, so the acceleration structure delivers a speed-up in excess of 100× while leaving the rest of the code unchanged.

5 Observed behaviour and correctness evidence

The mirror sphere exhibits crisp highlights from both the point light and the coloured walls, indicating that the reflection vector is computed correctly and that gamma is not applied twice. The solid-glass sphere distorts the background geometry and shows subtle reflections that fade

toward grazing angles, matching the Fresnel equations. The hollow sphere produces two concentric caustic-like rims: the outer one is the entry refraction; the inner one is caused by the ray leaving the inner cavity. Because the renderer includes only direct lighting, these caustics do not appear on the ground, which is expected.

In the second scene the cat's silhouette is smooth and shadow boundaries fall precisely where the mesh blocks the single light. Surfaces that face the light directly appear brighter; those turned away fall into self-shadow and show no light leaks, which demonstrates that the Möller–Trumbore routine respects back-face hits and that secondary shadow rays work with the same epsilon offsets as the primary rays.

Both scenes respect energy conservation in the sense that reflected and refracted weights sum to one and diffuse reflectance never exceeds unity. No pixel shows values above 1 prior to gamma compression, so quantisation to eight bits subjects only the dark tones to rounding error.

6 Closing remarks

The twin programmes trace primary rays, secondary reflections and refractions through both analytic and mesh geometry, calculate direct lighting with ray-cast shadows, apply gamma-correct output and exploit a BVH to keep the triangle count manageable. They compile with a modern C++ compiler and a single OpenMP switch, run entirely on the CPU, require no external libraries beyond the two stb headers, and reproduce the demonstrator images rapidly enough for iterative development. In short, they meet the objectives of the assignment while following the classical structure found in production renderers albeit in miniature form and illustrate how the same mathematical core scales from half a dozen spheres to hundreds of thousands of triangles simply by adding an efficient spatial index.