Lecture 02: Definition of computer graphics: Foundational Models

Instructor: Mejbah Ahammad

Intelligence Academy

Contents

Foundational Models

1.1 Geometric Primitives (Points, Lines, Polygons)

Geometric primitives are the canonical building blocks of graphical models. A *point* has position only; a *line segment* adds connectivity between two points; a *polygon* is an ordered loop of segments that defines a planar region. Modern GPU pipelines standardize on *triangles* because they are always planar, support affine interpolation (e.g., barycentric coordinates for attributes such as normals, UVs, and colors), and map efficiently to hardware rasterization.

Key details. Polygon orientation (winding) encodes front–back faces for back-face culling. Triangle meshes store vertices V, faces F, and often per-vertex attributes; fans/strips reduce index bandwidth. Smoothing groups or per-face normals control shading continuity, while non-manifold edges and degenerate triangles should be avoided for robust rendering and simulation.

Figure 1: Primitive progression with direction: points → lines → polygons → triangles (GPU standard).

1.2 Curves and Surfaces (Bezier, B-Splines, NURBS)

Smooth shapes are modeled with *parametric* functions. A *Bézier* curve is a weighted sum of control points using Bernstein polynomials; it provides global control and exact endpoint interpolation. *B-Splines* generalize Bézier with a knot vector and basis functions that offer *local support*, enabling edits without disturbing the whole shape. *NURBS* (Non-Uniform Rational B-Splines) add weights to model exact conics and CAD-grade surfaces.

Real-time pipelines typically *tessellate* curves and surfaces to adaptive triangle patches. Surface normals derive from parametric partials, and tessellation density is driven by screen-space error, curvature, or displacement bounds to balance quality and performance.

1.3 Volume and Voxel Representations

Volumetric models represent functions in 3D, commonly a scalar field $f(\mathbf{x})$ sampled on a *voxel* grid (dense or sparse). This is essential for medical data (CT/MRI), fluids, clouds, and fields where interior structure matters.

Two standard routes are (1) *ray marching*/compositing through $f(\mathbf{x})$ for direct volume rendering and (2) *iso-surface* extraction (e.g., Marching Cubes) to convert $f(\mathbf{x}) = c$ into a polygonal mesh. Gradients of f provide

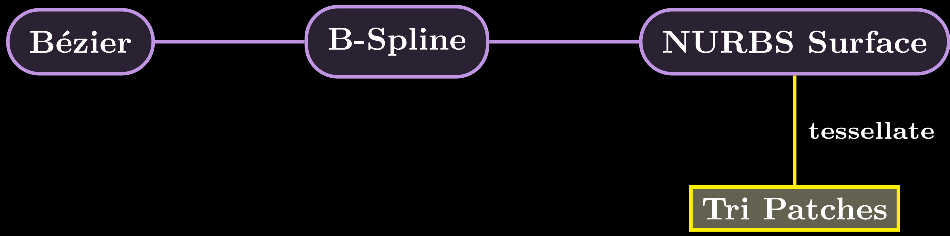

Figure 2: Parametric families with direction: curves → NURBS → adaptive tessellation into triangle patches.

normals for shading. Sparse structures (octrees, OpenVDB-style B-trees, or TSDF fusion) manage memory and accelerate queries.

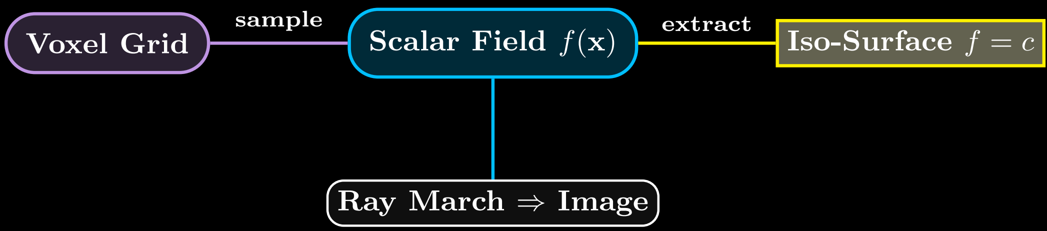

Figure 3: Volume pathways: voxels → $f(\mathbf{x})$ with options to ray-march (image) or extract iso-surfaces (mesh).

1.4 Implicit vs. Explicit Models

Explicit models. An explicit surface stores geometry directly—typically a triangle mesh with vertices V, faces F, and adjacency for operations such as smoothing, remeshing, and collision detection. Advantages include precise control over topology and straightforward GPU rendering; drawbacks are difficulty with robust boolean operations and smooth blending unless remeshed.

Implicit models. An implicit surface is the zero set $\{\mathbf{x} \mid F(\mathbf{x}) = 0\}$ of a function such as a signed distance field (SDF). Implicits excel at *CSG*, fillets, and smooth blends, and can be sampled at any resolution. Rendering requires root-finding (ray–surface intersection) or meshing via iso-surfacing, and fine details demand adequate sampling.

Conversion. Mesh ↔ field conversions are common: surface reconstruction (e.g., Poisson) builds F from points/meshes, while Marching Cubes extracts meshes from F for raster pipelines.

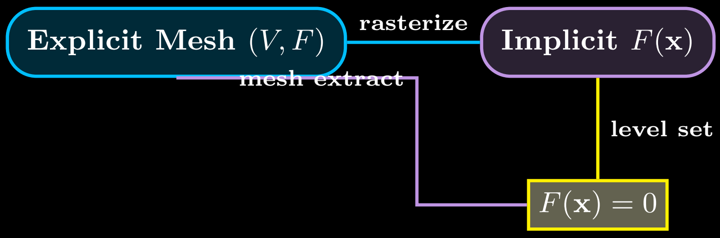

Figure 4: Representation trade-offs: explicit meshes vs. implicit functions with bidirectional conversion.

1.5 Scene Graphs and Hierarchies

Structure and purpose. A scene graph organizes a scene as a tree/DAG of nodes: *Transform* nodes encode local frames; *Geometry*, *Light*, and *Camera* nodes are leaves or subtrees. World transforms are composed top-down, enabling instancing, level-of-detail, and efficient culling via hierarchical bounds.

Traversal. Typical passes perform update and draw traversals (pre-/post-order). Dirty flags minimize recomputation of derived matrices and bounding volumes. Instancing shares geometry buffers across many transforms; visibility queries (frustum/occlusion) prune subtrees for performance.

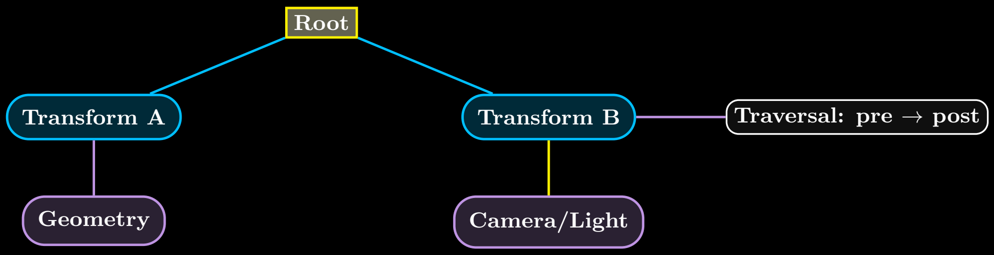

Figure 5: Scene graph hierarchy with directional composition and traversal hints.