Introduction

Computer graphics refers to the creation, manipulation and storage of visual content (pictures, drawings, diagrams, 3D models, etc.)
Technically, from modeling to animations/renderings to displaying on screens or other devices. **3D model → Render → Display**
Practically: movies, games, VR, art, advertising, etc.
CG vs CV: Models of 3D world (self-loop: 3D Visual Computing) & Pictures (self-loop: Digital Image Processing)
CG: left to right; CV: right to left.

History

The first graphic images are created by Ben Laposky in **1950** using an oscilloscope to generate waveform artwork produced by manipulating the analog electronic beams.
1960s: The term "Computer Graphics" was coined by William Fetter. World first video games — Space War, Tennis for Two, Pong. Theoretical work on computer graphics techniques.
1970s: Practical implementation of computer graphics techniques (diffuse lighting; specular lighting; texture; Z-buffer; anti-aliasing)
1980s: Global illumination and photorealism; Modernization and commercialization of computer graphics; Development of computer graphics systems; Computer graphics based movies such as Star Wars; Golden era of video games — Atari, Nintendo, Sega.
2000s: 3D modeling on mass scale for commercial use; 3D video games invented; Adoption for TV commercials, cinema; GPUs and sophisticated computation hardwares.

Geometry

Curve

EXAMPLES: Camera Paths; Animation Curves (follow the curve to move); Vector Fonts.
Explicit Curve: $y = f(x)$; **Implicit Curve**: $f(x, y) = 0$; **Parametrized Curve**: $\gamma(t) = \big(x(t), y(t)\big)$.
1D curves in 2D: **Polylines** (D: not smooth);
Smooth curve: **Spline** (D: harder to specify) (user specifies control points; **interpolation** [A: logical, go through all points; D: unstable] or **approximation** [A: convenient]).
Bezier Curves: $\boldsymbol{b}^n(t) = \sum_{i=0}^{n} \boldsymbol{b}_i B_i^n(t)$, $t \in [0,1]$. $B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}$ is i-**th Bernstein poly of degree** n.
de Casteljau Alg.: insert points using linear interpolation in every adjacent pair of points. Repeat n times.
Piecewise Bezier Curves: chain many **cubic** Bezier curves. Ensure $\mathbb{C}^1$ continuity: need $\boldsymbol{a}_n = \boldsymbol{b}_0 = (\boldsymbol{a}_{n-1} + \boldsymbol{b}_1)/2$.
General Spline Formulation: $Q(t) = \boldsymbol{GBT}(t)$. *Geometry(control points), Spline Basis(define type), Power Basis(monomials).*

Surface

Parametrization: f that maps $U \subseteq \mathbb{R}^2$ into $\mathbb{R}^n$. $f(U)$ is the *image*.
Bezier Surface: $\boldsymbol{s}(u, v) = \sum_{0 \le i \le m, 0 \le j \le n} \boldsymbol{p}_{i,j} B_i^m(u) B_j^n(v)$. **Separable 1D de Casteljau Alg.**: Calculate dimension by dimension.

Explicit Geometric Representation

3D Representation: **Rasterized Form** (regular grid) **& Geometric Form** (irregular); **Origin dependent & Application dependent**
EXAMPLES: **Multiview 3D rep.** (Unstructured Image → Scene graph → sparse model → dense model. D: no explicit geometry A: can reconstruct. Viewpoint matters a lot);
Depth map (D: incomplete 3D; resolution issue; object properties matter; need camera information for true 3D).
Volumetric rep. (D: obvious artifacts; storage & computation cost).
Point Cloud. (A: Simple to understand; compact to store; generally easy to build algorithms. D: No orientation) Points with orientation: **surfels**. Build from real world: **3D scanning** (striped data, need multiple views. *Challenge:* resolution, occlusion, noise).
Uniform Sampling from surface (A: easy to implement. D: irregularly spaced sampling).
Farthest Point Sampling (D: NP-hard A: have good approx.).
Triangle Mesh: polygonal mesh where every face is a triangle; simplifies data structures, rendering, algorithms; each face planar and convex; any polygon can be triangulated.
Mesh. (Use to visualize 3D; ground truth for ML) *Manifold Mesh*: each edge incident to ≤ 2 faces; faces incident to a vertex form a closed or open fan.
BAD meshes: "Triangle Soup"; open boundaries; self-intersection; non-manifold; ambiguous face orientation; double surfaces; non-uniform areas & angles. (non-manifold)
Need to be *Watertight* (Alg: voxelize surface → extract exterior faces → projection-based optimization)
Mesh Reconstruction (Cleaning; repairing; remeshing):
Explicit Method: three points form a triangle if a ball of radius ρ touches them without containing any other points;
Implicit Method: estimate implicit field function from data, extract zero iso-surface.
Data Structure for Surfaces: Geometry (position); topology; attributes (normal, color, texture)
EXP: face list (STL in CAD, no connectivity information); indexed face set (OBJ, counter-clockwise order for normal computation).
Loop Subdivision: for two adjacent triangles with edge AB in common, $P \leftarrow 3/8(A + B) + 1/8(C + D)$ is the **new point**.
For **old point** Q, $Q \leftarrow (1 - nu) \cdot Q + u \cdot \sum_{Q' neighbor of Q} Q'$
($u = 3/16$ if $n = 3$, $3/(8n)$ otherwise; n = #neighbors).
Catmull-Clark Subdivision (For quad mesh): face point $f = (v_1 + v_2 + v_3 + v_4)/4$; edge point $e = (v_1 + v_2 + f_1 + f_2)/4$; vertex point $v \leftarrow (\sum_4 f_i + 2\sum_4 e_i + 4v)/16$.
Mesh Simplification: By edge collapsing. Minimize *quadric error*: $\min_p \sum_i dist(q_i, p)$, p is the selected point, $\{q_i\}$ are triangles adjacent to two original points. Iterative collapse edge with min err.
Mesh Deformation: Consider both *Objective* and *Constraints*. **Local** deformation energy $E(v_i) = \min_{R_i} \sum_{j \in N(i)} ||\,(v_i' - v_j') - R_i(v_i - v_j)||^2$, R_i is rotation. Stretching (length change) & Bending (angle change) increases E. Translation and rotation doesn't change.
ARAP Deformation: minimize **total** deformation energy $E(V') = \min_{v'} \sum_i E(v_i)$, such that $v'_j = c_j$ (c_j is any control point).
Free-Form Deformation: displace each vertex by $\boldsymbol{d}_{ijk}$, then $\boldsymbol{d}(x, y, z) = \sum_{i,j,k} B_i(x) B_j(y) B_k(z) \boldsymbol{d}_{ijk}$.

Implicit Geometric Representation

Obtain more complex shapes by doing $A \cap B, A \cup B, A \setminus B$.
Distance function: minimum dist. to object. Use it to smoothly combine two objects by tuning threshold.
Level Set Method: store grid of values approximating function, such that interpolated values $f(x) = 0$ represents the surface.

Transformation

Rotation, scale, reflection, shear: LINEAR; translation: NOT LINEAR. These are all AFFINE transformations.
Shear, scale is not EUCLIDEAN ($|f(x) - f(y)| = |x - y|$).
Represent 3D transformation as 4×4 matrices: using **homogeneous coordinate** $(x, y, z, w) \sim (x/w, y/w, z/w, 1)$.
Set of Rotations: $SO(n) = \{R \in \mathbb{R}^{n \times n} : det(R) = 1, RR^T = I\}$. $SO(3)$ has 3 DoF. Topology is different from $(-l, l)^n$, so difficult to parameterize in NN!
Euler Angle: $R = R_z(\gamma) R_y(\beta) R_x(\alpha)$, $R_x(\alpha) = \begin{pmatrix} cos\gamma & -sin\gamma & 0 \\ sin\gamma & cos\gamma & 0 \\ 0 & 0 & 1 \end{pmatrix}$.
Gimbal Lock: when $\beta = \pi/2$, $R = \begin{pmatrix} 0 & 0 & 1 \\ sin(\alpha+\gamma) & cos(\alpha+\gamma) & 0 \\ -cos(\alpha+\gamma) & sin(\alpha+\gamma) & 1 \end{pmatrix}$.
Euler's Thm: $\forall R \in SO(3), \exists \left|\left|\hat{\omega}\right|\right| = 1, \theta$, such that $R = Rot\big(\hat{\omega}, \theta\big)$.
(rotate about axis $\hat{\omega}$ for angle θ)
Let $[\omega] = \begin{pmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{pmatrix}$, we have $a \times b = [a]b$.
$Rot\big(\hat{\omega}, \theta\big) = e^{[\hat{\omega}]\theta}$ (**Exponential Coordinate**). Use **Rodrigues Formula** $\left[\hat{\omega}\right]^3 = -\left[\hat{\omega}\right], e^{\left[\hat{\omega}\theta\right]} = I + \left[\hat{\omega}\right] sin\theta + \left[\hat{\omega}\right]^2 (1 - cos\theta)$.
Restricting $\theta \in (0, \pi]$, $tr(R) \neq -1$, then $\theta = arccos\frac{tr(R)-1}{2}$, $\left[\hat{\omega}\right] = \frac{R - R^T}{2sin\theta}$. When $\theta = 0$, we have $R = I$. Thus, the distance between two $R_1, R_2 \in SO(3)$ is $\theta(R_2 R_1^T) = arccos\frac{tr(R_2 R_1^T)-1}{2}$.
Quaternion: $q = a + b\boldsymbol{i} + c\boldsymbol{j} + d\boldsymbol{k}$. Rotate $\boldsymbol{x}$ by q: augment $\boldsymbol{x}$ to $(\boldsymbol{0}, \boldsymbol{x})$ then $q\boldsymbol{x}q^{-1}$.
Exponential coordinate → Quaternion: $q = \left[cos\theta/2, sin\theta/2 \cdot \hat{\omega}\right]$.
Rotation ← Quaternion: $q = (w, x, y, z)$ where $||q|| = 1$, then $R(q) = E(q)G(q)^T$ where $E(q) = [-v, wI + [v]]$, $G(q) = [-v, wI - [v]]$, explicitly

$$R(q) = \begin{pmatrix} 1-2y^2-2z^2 & 2xy-2zw & 2xz+2yw \\ 2xy+2zw & 1-2x^2-2z^2 & 2yz-2xw \\ 2xz-2yw & 2yz+2xw & 1-2x^2-2y^2 \end{pmatrix}.$$

Rotation to Quaternion: **convert to** $\big(\hat{\omega}, \theta\big)$ **first**!

Projection

View/Camera Transformation: Transform the camera to origin, up at Y, look at -Z, $M_{\text{view}} = R_{view} T_{view}$, $T_{view} = \begin{pmatrix} 1 & 0 & 0 & -x_e \\ 0 & 1 & 0 & -y_e \\ 0 & 0 & 1 & -z_e \\ 0 & 0 & 0 & 1 \end{pmatrix}$, $R_{view}^{-1} = \begin{pmatrix} x_{\hat{g}\times\hat{t}} & x_t & x_{-g} & 0 \\ y_{\hat{g}\times\hat{t}} & y_t & y_{-g} & 0 \\ z_{\hat{g}\times\hat{t}} & z_t & z_{-g} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
orthographic projection: $M_{ortho} = M_1 M_2$: M_1 is diagonal matrix scaling each dimension into $[-1,1]$ $(r, t, n) \rightarrow (1,1,1)$, while M_2 is translation matrix placing center into $(0,0,0)$. *Note: $n > f$, meaning we look at $-z$ direction!*
perspective projection.: $M_{persp} = M_{ortho} M_{persp \rightarrow ortho}$, where

$$M_{persp \rightarrow ortho} = \begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -nf \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

To specify l, r, b, t in the near plane, use fovY (field of view) and aspect_ratio = width / height. $tan(fovY/2) = t/|n|$.
Sequence of Transforms: modeling → camera → projection → viewport (3D to 2D).

Learning + Geometry

Geometric Datasets

ShapeNet: A large-scale dataset of 3D shapes

PartNet: Fine-grained semantic part dataset
ScanNet (++): Large-scale indoor scene dataset
AI + Geometry: S shape, c label or condition, $P(c|S)$ shape analysis (learning shape descriptors, shape classification, segmentation, view estimation), $P(S|c)$ shape synthesis (learning shape priors, shape generation, completion and reconstruction)

Data-driven shape analysis

Multi-View CNN: CNN for each view -> view pooling -> CNN
A: Leverage vast literature of image classification and pre-trained models D: Need projection, bad for noisy or incomplete input
Voxelization: represent the occupancy of regular 3D grid
Issue: complexity (4D kernel), information loss in voxelization, sparsity of 3D shape
Idea: 1. Learn to project 3D shape to 2D + CNN
2. Only occupied grids and constrain computation near the surface
3. Sparse convolution: only compute on occupied grids
Octree: recursively partition 3D space into 8 equal parts, hash table for neighbor search
Point cloud: unordered set of points, no topology, no connectivity permutation invariance for feature learning
vanilla **PointNet**: feature extractor h-> symmetric function g -> MLP

$$f(\{x_1, x_2, \dots, x_n\}) = \gamma \odot g\big(h(x_1), h(x_2), \dots, h(x_n)\big)$$

Limitation: no local context for each point, hard to generalize to unseen configurations (3D CNN hierarchical feature learning)
PointNet++: recursively apply PointNet to local region, hierarchical feature learning, local translation invariance, permutation invariance
GNN on point cloud: neighbor -> edge, point -> node
Issue: not sample invariant (different sampling -> different graph) Sol: estimate the continuous kernel and point density for continuous conv (Monte Carlo conv)
Isometric Invariance Recognition: apply conv in spectral domain of Laplacian matrix.
If the shapes are not isometric, spectral domains are not aligned.
Function bases derived by Laplacian operator, geometry dependent

Shape synthesis

Predict depth with loss

$$L(y, y^*) = \sum_i \left|\left|\log y_i - \log y_i^* + \alpha(y, y^*)\right|\right|^2$$

$$\alpha(y, y^*) = \frac{1}{n}\sum_i \log y_i^* - \log y_i$$

Object-centric 3D: decide model and objectives -> collecting training data -> learn predictor
Predict Volumetric 3D: Image -> latent -> D^3 volume
Octree: at each level, predict 3-way label (empty, occupied, mixed), hierarchical decode voxels
Answer occupancy query of an arbitrary 3D point instead of decoding a volumetric representation
Predict Point Cloud:
Chamfer Distance: $L_{CD}(X, Y) = \sum_{x \in X} \min_{y \in Y} ||x - y||^2 + \sum_{y \in Y} \min_{x \in X} ||x - y||^2$ 2. Parametric shape: Latent shape representation + sampled points on planes -> generated 3D point
Predict Mesh 1. predict deformation instead of absolute position 2. Training objective: loss on point cloud of predicted mesh and ground truth mesh 3. Regularization: smoothness (surface normal of adjacent faces should align)
First predict a Volumetric 3D representation, then deform from a meshified voxel representation
Infering 3D shape from 2D image

Rasterization

Frame Buffer: Memory for a Raster Display
LCD Pixel: principle: block or transmit light by twisting polarization, illumination from backlight
Triangles: most basic polygon; guaranteed to be planar, well-defined interior, well-defined method for interpolating values at vertices
Rasterizing a triangle: for each pixel (x, y), check if $(x + 0.5, y + 0.5)$ is inside the triangle (hierarchical rasterization since triangle is convex)
Hierarchical Rasterization, Incremental Triangle Traversal
Sampling Artifacts in CG: Jaggies (sampling in space), Moire (undersampling images), Wagon wheel effect (sampling in time)
Aliasing: signals change too fast for the sampling rate
Undersampling creates frequency aliases because high-frequency signal is insufficiently sampled, samples erroneously appear to be from a low-frequency signal
Reducing Aliasing Error: increase sampling rate / Antialiasing (i.e. filtering out high frequencies before sampling)
Idea: Blurring(pre-filtering) before sampling, get rid of certain frequency contents (Convolution Theorem: convolution in spatial domain = point-wise multiplication in frequency domain)
Sampling = Repeating Frequency Contents
Anti-aliasing: 1. Average values in pixel area 2. Multi-sampling: sample multiple times in a pixel area 3. Super-sampling: render at higher resolution and down-sample
Z-buffer: for each pixel, store the depth value of the closest object

Shading

Diffuse Reflection: Surface color is the same for all viewing directions, $k_d$is diffuse coefficient

$$L_d = k_d \frac{I}{r^2}\max(0, n \cdot l).$$

Specular Reflection: $h = \frac{l+v}{||l+v||}$, bright near mirror reflect direction

$$L_s = k_s \frac{I}{r^2}\max(0, n \cdot h)^p.$$

Ambient Term: Fake, approximate environment light

$$L_a = k_a I_a.$$

Blinn-Phong Model: Diffuse + Specular + Ambient
Flat Shading: One norm vector on surface, bad for **smooth surface**
Gouraud Shading: interpolate colors from vertices across triangle.
Phong Shading: Interpolate normal vectors across each triangle. Full shading at pixel. **Not Blinn-Phong Reflectance.** Per-pixel normal.
Simple: each vertex has normal vector $N_v = \frac{\sum N_i}{||\sum N_i||}$,
Barycentric interpolation of vertex normal.
Graphics Pipeline: Vertex Processing (Model, view, project transforms) -> Triangle Processing -> Rasterization (sampling triangle coverage) -> Fragment Processing(Z-buffer, shading, texture mapping) -> Framebuffer Operations.
Graphics Pipeline Implementation: GPU (heterogeneous, multi-core)

Texture Mapping

Goal: flatten 3D object onto 2D UV coordinates. For each vertex, find coordinates U, V such that distortion is minimized.
Divide surface into triangles, map texture to each triangle.
Barycentric Parametrization: Fix (u, v) of the boundary, solve

$$v_i = \frac{1}{valence(i)}\sum_{j \in N(i)} v_j.$$

Barycentric coordinates $(x, y) = \alpha A + \beta B + \gamma C$,

$$\alpha = \frac{-(x - x_B)(y_C - y_B) + (y - y_B)(x_C - x_B)}{-(x_A - x_B)(y_C - y_B) + (y_A - y_B)(x_C - x_B)}.$$

Texture queries

Simple Texture Mapping: Diffuse color, evaluate (u, v) for each (x, y)
Point sampling causes **Jaggies** and **Moiré**.
Bilinear Interpolation: give good results at reasonable costs.
Moiré caused by taking a point in range instead of mean.
Supersampling work: high quality, but costly. When highly minified, many texels in pixel footprint, signal freq too large in a pixel.
Mipmap: Mip Hierarchy means precomputed downsampled versions of the texture. Pixel range doubled each level.

$$D = \log_2 \max\left(\sqrt{\left(\frac{du}{dx}\right)^2 + \left(\frac{dv}{dx}\right)^2}, \sqrt{\left(\frac{du}{dy}\right)^2 + \left(\frac{dv}{dy}\right)^2}\right).$$

Linear interpolation between bilinear on two integer levels. **Overblur** on anisotropic scenarios.
Other filtering methods: **Ripmaps and summed area tables** (Can look up axis-aligned rectangular zones, Diagonal footprints still a problem), **EWA filtering** (Use multiple lookups, weighted average, can handle irregular footprints, Mipmap hierarchy still helps).

Applications of textures

Environment Mapping: Place a small reflective chrome sphere (a light probe) somewhere in the world. Photograph it to measure/record the intensity of light incoming from all directions.
Cube Map: Cut the sphere into 6 faces of a cube.
Bump Mapping: Perturb the normal of the surface per pixel. $n = (-dp/du, -dp/dv, 1)$, in local coordinate
Displacement mapping: Move the vertices of the surface. Quality depends on shadow map resolution.

Shadow mapping

Image-space algorithm: no knowledge of scene's geometry during shadow computation; must deal with aliasing artifacts
Depth map from view of light and view of eye. Decide in shadow by matching two depths. **No global effect/soft shadows.**

Ray Tracing I

Ray casting

Three ideas about light rays
1. Light travels in straight lines
2. Light rays do not "collide" with each other if they cross
3. Light rays travel from the light sources to the eye

Whitted-Style ray tracing

Recursive ray tracing: split the ray into reflection and refraction rays.

Accelerating ray tracing

Bounding Volumes: Axis-Aligned Bounding Box (1 sub, 1 div)
Uniform Spatial Partitions: Preprocess:1. Find bounding box; 2. Create grid; 3. Store each object in overlapping cells. Step through grids in ray transversal order. (Grid resolution: #cells = C * #objs) work well on objects distributed evenly in size and space
Oct-Tree: Top-down approach (intersect root and find intersected child recursively); Bottom-up approach (find first intersected leaf, use neighbor finding to find next voxels)
BSP-Tree: space partitioned into half spaces, all bins convex, difficult to build, traversal not bad
KD-Tree: Preprocess each time split a cell into two. **No objects are stored in internal nodes.** For each node, when searching for intersection, split into small subtasks for each child node.
Bounding Volume Hierarchy (BVH): Similar to KD-Tree, but use bounding boxes to represent the cells. Each box bound a set of objects. A: easy to construct, easy to traverse, binary tree. D: difficult to choose a good split, poor split result in minimal spatial pruning. (Heuristic: choose longest axis / split node at location of median object)