

3D Object Representations

Róbert Bohdal

`robert.bohdal@fmph.uniba.sk`

`https://flurry.dg.fmph.uniba.sk/webog/sk/bohdal-teaching`

Department of Algebra and Geometry
Faculty of Mathematics, Physics and Informatics
Comenius University Bratislava

Computer Graphics (1)
lectures 3 and 4



Contents

- 1 Object representations and modeling methods
- 2 Polygon meshes
- 3 Surfaces created by sweeping and templating
- 4 Fractal objects generated by Brownian midpoint displacement
- 5 Objects created by reconstructing real-world data
- 6 Splines
- 7 CSG
- 8 Recursive space subdivision
- 9 Objects defined by implicit functions
- 10 Metaballs



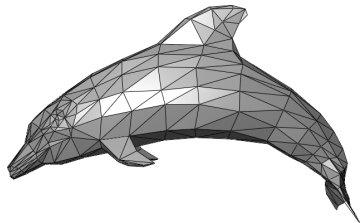
Object representations and modeling methods

- Boundary representation – objects are defined by their boundary (e.g., a spline surface)
- Volume representation – objects are composed of volumetric elements (e.g., voxels)
- Polygon meshes
- Objects created by sweeping and templating
- Objects generated by procedural methods (e.g., fractal methods)
- Objects defined by point clouds
- Objects reconstructed using photogrammetry
- Splines
- CSG trees (Constructive Solid Geometry trees)
- Space subdivision
- Implicit function representations (F-rep)
- Metaballs (blobs)



Polygon meshes

- A very simple representation, although it only approximates the actual surface of a solid
- Currently the most widely used representation
- Supports fast, optimized algorithms (hardware acceleration, parallelization)
- Requires a large amount of data for detailed representation
- Shape modification usually requires recomputing many values (vertices)



en.wikipedia.org/wiki/Polygon_mesh



Polygon meshes

Definition (Regular polygon mesh)

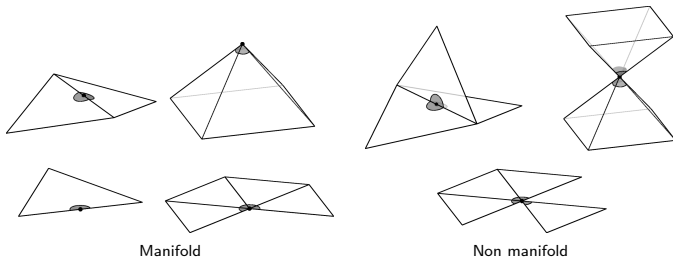
A regular polygon mesh is a triple $\mathcal{M} = (\mathcal{V}, \mathcal{E}, \mathcal{F})$, where $\mathcal{V} \subset \mathbb{R}^3$ is a finite set of vertices, $\mathcal{E} \subseteq \{u, v \mid u, v \in \mathcal{V}, u \neq v\}$ is a set of undirected edges, and $\mathcal{F}$ is a finite set of simple, nondegenerate planar polygons whose vertices belong to $\mathcal{V}$ and whose edges belong to $\mathcal{E}$. Furthermore:

1. every vertex and every edge belongs to at least one face,
 2. for any two distinct faces $F_1, F_2 \in \mathcal{F}$, their intersection is either empty, a complete common edge, or a common vertex.
- Triangles are the most commonly used polygons. Every n -gon ($n \geq 3$) can be triangulated, and the vertices of a triangle are coplanar
 - Often used together with texture mapping (texture coordinates are defined at the vertices)
- Properties:
- A polygon mesh is connected if its graph is connected, where the graph vertices represent the mesh vertices and the graph edges represent the mesh edges
 - The boundary of a polygon mesh is the union of edges that belong to only one face
 - A polygon mesh is closed if it has no boundary



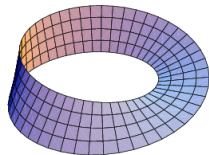
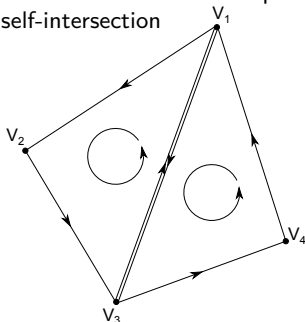
The 2-manifold property

- A polygon mesh is a 2-manifold if every edge is incident to one or two faces and the faces incident to a vertex form a closed or open, but uninterrupted, fan

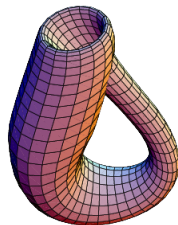


Polygon meshes and orientation

- Let a polygon mesh be a 2-manifold. It is orientable if an orientation can be chosen for every polygon such that every pair of adjacent polygons is oriented consistently (edges shared by adjacent faces have opposite orientations within those faces)
- The Möbius strip is an example of a nonorientable polygon mesh. The Klein bottle is an abstract nonorientable 2-manifold whose representation in $\mathbb{R}^3$ has a self-intersection



commons.wikimedia.org/wiki/File:MöbiusStrip-01.svg



commons.wikimedia.org/wiki/File:KleinBottle-01.png

Euler characteristic χ

- A number that describes the shape or structure of an object represented by a polygon mesh, independently of any “regular deformation”
- $\chi = V - E + F$, where V is the number of vertices, E is the number of edges, and F is the number of faces
- For convex polyhedra (cube, tetrahedron, etc.) in $\mathbb{E}^3$, this number is always 2
- For orientable polygon meshes with G holes and possibly B boundary components (i.e., meshes that are not closed), $\chi = 2(1 - G) - B$
- A torus (a polygon mesh with one hole) has $G = 1$, and therefore $\chi = 0$
- The Möbius strip and the Klein bottle (nonorientable polygon meshes) have $\chi = 0$



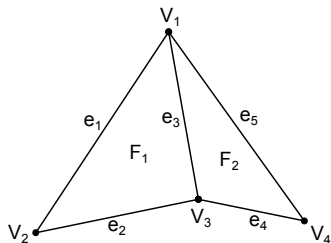
Data structures for polygon mesh representation

- Simple (classical) representation
- Winged-edge representation
- Half-edge representation (DCEL - Doubly Connected Edge List)



Simple representation

- Uses tables of:
 vertices $\mathbb{V} = \{V_i; V_i = (x_i, y_i, z_i)\}$,
 edges $\mathbb{E} = \{e_j; e_j = (V_{j_1}, V_{j_2}); j_1 \neq j_2\}$
 faces $\mathbb{F} = \{F_k; F_k = (e_{k_1}, \dots, e_{k_m}), k_1 \neq \dots \neq k_m\}$
- The face table $\mathbb{F}$ may contain a list of vertices instead of a list of edges



Vertex table			
V_1	x_1	y_1	z_1
V_2	x_2	y_2	z_2
V_3	x_3	y_3	z_3
V_4	x_4	y_4	z_4

Edge table		
e_1	V_1	V_2
e_2	V_2	V_3
e_3	V_3	V_1
e_4	V_3	V_4
e_5	V_4	V_1

Face table			
F_1	e_1	e_2	e_3
F_2	e_3	e_4	e_5

- $\oplus$: Simple implementation
- $\ominus$: Complicated neighborhood queries, including adjacent faces and edges incident to a vertex, etc.

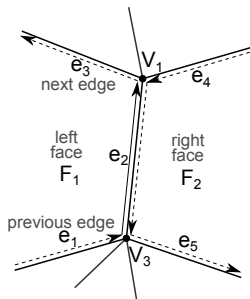
Complex mesh modifications are cumbersome because neighborhood information is missing

The face table requires a dynamic data structure because it does not have a fixed number of entries



Winged-edge representation

- Uses a vertex table containing vertex coordinates and one incident edge
- The edge list contains eight entries for each edge: references to its start and end vertices, its left and right faces, and the previous and next edges in both the left-face traversal and the right-face traversal
- The face list contains only a reference to one incident edge for each face
- Each edge is stored in the table as an oriented line segment, but this orientation is independent of the edge orientation within a particular mesh face (polygon)

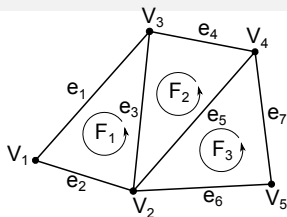


Edge table

edge	vertex		face		left traversal		right traversal	
ID	start	end	left	right	prev.	next	prev.	next
e ₂	V ₃	V ₁	F ₁	F ₂	e ₁	e ₃	e ₄	e ₅



Winged-edge representation



Vertex table

V_1	x_1	y_1	z_1	e_2
V_2	x_2	y_2	z_2	e_6
V_3	x_3	y_3	z_3	e_3
V_4	x_4	y_4	z_4	e_4
V_5	x_5	y_5	z_5	e_7

Edge table

edge	vertex		face	left traversal		right traversal		
ID	start	end	left	right	prev.	next	prev.	next
e_1	V_1	V_3	F_1				e_3	e_2
e_2	V_1	V_2	F_1		e_1	e_3		
e_3	V_2	V_3	F_1	F_2	e_2	e_1	e_4	e_5
e_4	V_3	V_4	F_2				e_5	e_3
e_5	V_2	V_4	F_2	F_3	e_3	e_4	e_7	e_6
e_6	V_2	V_5	F_3		e_5	e_7		
e_7	V_4	V_5	F_3				e_6	e_5

Face table

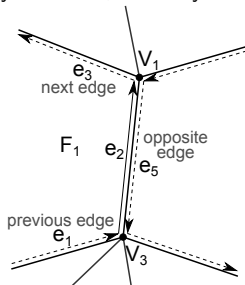
F_1	e_1
F_2	e_3
F_3	e_5

- $\oplus$: Provides fast access to neighborhoods; incident edges of a given vertex and all faces containing that vertex can be found quickly
- Each table has a fixed number of entries



Half-edge representation

- Contains a vertex table in which every vertex stores its coordinates and a reference to one of its **outgoing** half-edges
- The half-edge table contains references to the origin vertex of the current half-edge, the face to its left, its opposite (twin) half-edge, and the previous and next half-edges in the left face
- Each half-edge belongs to only one face, so every interior edge is represented twice

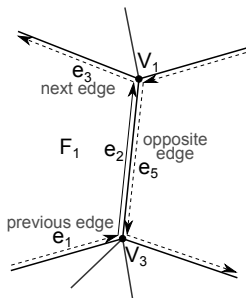


- $\oplus$: Provides fast access to neighborhoods
Supports an efficient representation of dynamic meshes whose topology changes over time
- $\ominus$: This structure can represent only orientable surfaces
It requires more memory



Half-edge representation

- Each half-edge stores references to its origin vertex, incident face, twin half-edge, and the previous and next half-edges
- In this example, e_2 starts at V_3 , belongs to face F_1 , and has e_5 as its twin
- Along the boundary of F_1 , e_1 precedes e_2 and e_3 follows it. Therefore, the destination of e_2 is V_1 , the origin of e_3

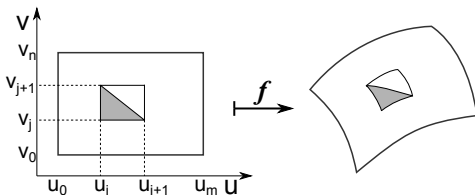


Half-edge table

half-edge	vertex	face	opposite	left-face traversal	
ID	origin	left	twin	prev.	next
e_2	V_3	F_1	e_5	e_1	e_3

Surfaces created by sweeping and templating

- These are usually parametric surfaces defined by the vector-valued function $\mathbf{f}(u, v) = [f_x(u, v), f_y(u, v), f_z(u, v)] \in \mathbb{R}^3$ with parameters u and v , where $u \in [u_0, u_m]$ and $v \in [v_0, v_n]$
- The vertices of a polygon mesh corresponding to this parametric surface are given by specific values $\mathbf{f}(u_i, v_j)$
- Individual faces consist of triangles $\mathbf{f}(u_i, v_j)$, $\mathbf{f}(u_{i+1}, v_j)$, $\mathbf{f}(u_i, v_{j+1})$ and $\mathbf{f}(u_{i+1}, v_{j+1})$, or quadrilaterals: $\mathbf{f}(u_i, v_j)$, $\mathbf{f}(u_{i+1}, v_j)$, $\mathbf{f}(u_{i+1}, v_{j+1})$, $\mathbf{f}(u_i, v_{j+1})$



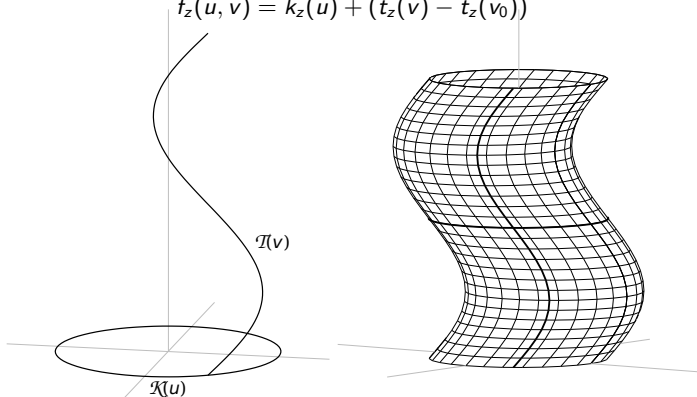
Translational surfaces

- They are formed by a generating curve $\mathcal{K}(u) = [k_x(u), k_y(u), k_z(u)]$ translated along a translation curve $\mathcal{T}(v) = [t_x(v), t_y(v), t_z(v)]$ starting at v_0 . The curve $\mathcal{T}(v)$ specifies the translation vector of the generating curve $\mathcal{K}(u)$
- The resulting translational surface $\mathbf{f}(u, v)$ is given by:

$$f_x(u, v) = k_x(u) + (t_x(v) - t_x(v_0))$$

$$f_y(u, v) = k_y(u) + (t_y(v) - t_y(v_0))$$

$$f_z(u, v) = k_z(u) + (t_z(v) - t_z(v_0))$$



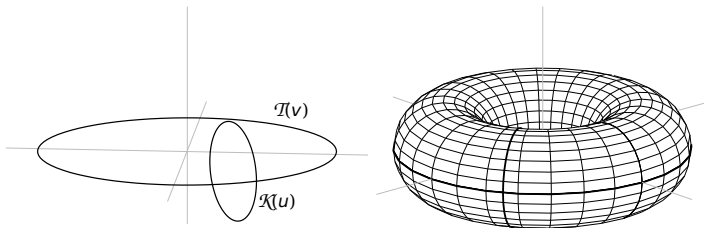
Surfaces of revolution

- They are most commonly generated by rotating a curve $\mathcal{K}(u) = [k_x(u), k_y(u), k_z(u)]$ about a given axis. Usually, a curve lying in the xz -plane, i.e., $k_y(u) = 0$, is rotated about the z -axis through an angle $v \in [0, 2\pi)$. Rotation about the z -axis transforms $[k_x(u), 0, k_z(u)]$ into $[k_x(u) \cdot \cos(v), k_x(u) \cdot \sin(v), k_z(u)]$.
- The surface of revolution $f(u, v)$ is therefore given by:

$$f_x(u, v) = k_x(u) \cdot \cos(v)$$

$$f_y(u, v) = k_x(u) \cdot \sin(v)$$

$$f_z(u, v) = k_z(u)$$

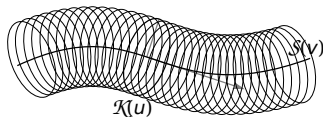


Note: A torus is generated by rotating the circle $k_x(u) = R + r \cdot \cos(u)$ and $k_z(u) = r \cdot \sin(u)$.



Surfaces generated by general sweeping

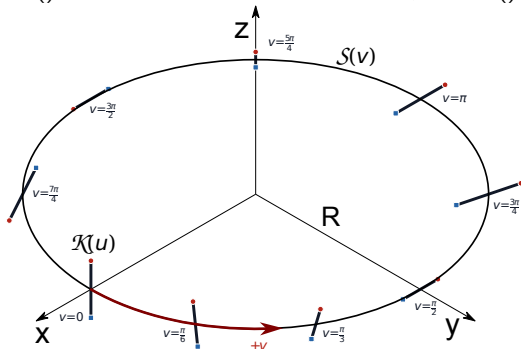
- These surfaces are generated by moving a generating curve $\mathcal{K}(u)$ along a sweep path $\mathcal{S}(v)$ so that, at each point, the generating curve is “perpendicular” to the tangent vector $\mathcal{S}'(v)$
- A roller-coaster car illustrates this motion: it travels along the track in the direction of $\mathcal{S}'(v)$, while the twist determines how the car rolls, or banks, about this tangent



- $\ominus$: Computing the orientation of $\mathcal{K}(u)$ as it moves along $\mathcal{S}(v)$ is difficult. A local frame must follow the sweep path, while the twist controls its rotation about the tangent vector $\mathcal{S}'(v_0)$ at the current point $v = v_0$

Möbius strip

- It is generated by first rotating the line segment $\mathcal{K}(u) = [R, 0, u]$ where $u \in [-a, a]$ through an angle $v/2$ about an axis parallel to the y -axis and passing through the segment midpoint. It is then rotated about the z -axis through an angle v , while its midpoint moves along the circle $\mathcal{S}(v)$. The parameter v belongs to the interval $[0, 2\pi)$.
- The parametric equations of this surface can be expressed by the matrix product $R_z(v) \cdot T_x(-R)^{-1} \cdot R_y(v/2) \cdot T_x(-R) \cdot X$, where the coordinates of the point $X[x, y, z]$ are replaced by the equations of the line segment $\mathcal{K}(u)$. $R_y()$ denotes a rotation matrix about the y -axis, $R_z()$ a rotation matrix about the z -axis, and $T_x()$ a translation matrix along the x -axis.



Example: derivation of the parametric equations of a Möbius strip

$$\begin{aligned}
 \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} &= \overbrace{\begin{pmatrix} 1 & 0 & 0 & R \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}}^{T_x(-R)^{-1}=T_x(R)} \cdot \overbrace{\begin{pmatrix} \cos(v/2) & 0 & \sin(v/2) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(v/2) & 0 & \cos(v/2) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}}^{R_y(v/2)} \cdot \overbrace{\begin{pmatrix} 1 & 0 & 0 & -R \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}}^{T_x(-R)} \cdot \overbrace{\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}}^x \\
 &= \begin{pmatrix} \sin(v/2)z + \cos(v/2)(x - R) + R \\ y \\ \cos(v/2)z - \sin(v/2)(x - R) \\ 1 \end{pmatrix} \\
 &= \begin{pmatrix} R + u \sin(v/2) \\ 0 \\ u \cos(v/2) \\ 1 \end{pmatrix} \quad (\text{we substituted } x = R, y = 0, z = u, \text{ i.e., the curve } \mathcal{K}(u) = [R, 0, u])
 \end{aligned}$$

$$\begin{pmatrix} x'' \\ y'' \\ z'' \\ 1 \end{pmatrix} = \overbrace{\begin{pmatrix} \cos(v) & -\sin(v) & 0 & 0 \\ \sin(v) & \cos(v) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}}^{R_z(v)} \cdot \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} (R + u \sin(v/2)) \cos(v) \\ (R + u \sin(v/2)) \sin(v) \\ u \cos(v/2) \\ 1 \end{pmatrix}$$

The Möbius strip can therefore be written as:

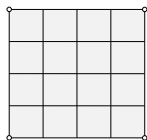
$$f(u, v) = ((R + u \sin(v/2)) \cos(v), (R + u \sin(v/2)) \sin(v), u \cos(v/2))^T,$$

where $u \in [-a, a]$ and $v \in [0, 2\pi]$.

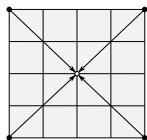


Fractal objects generated by Brownian midpoint displacement

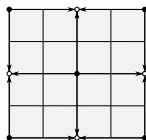
- They are usually generated using the diamond-square algorithm
- In the diamond step (diagonal neighbours), the height of the midpoint of each square is calculated as the average of its four corner points plus a random value
- In the square step (orthogonal neighbours), the height of the midpoint of each diamond is again calculated as the average of its corner points plus a random value



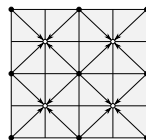
initialization



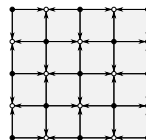
1st diamond step



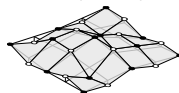
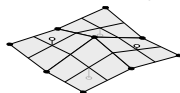
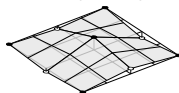
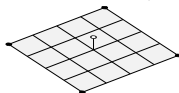
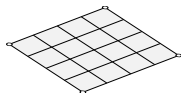
1st square step



2nd diamond step



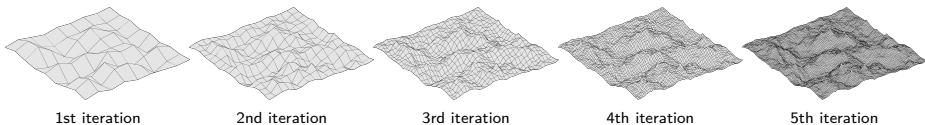
2nd square step



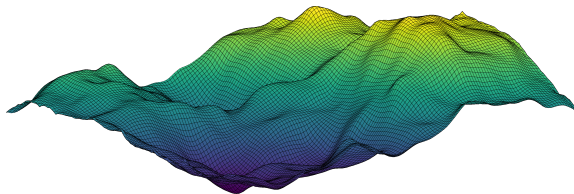
- At iteration i , the random displacement amplitude is scaled by the factor $\left(\frac{1}{2-h}\right)^{i \cdot s}$
 - $h \in [-1, 0.1]$ is the roughness parameter. Smaller values make the displacements decay faster and produce smoother terrain, while larger values produce rougher terrain
 - $s \in [1, 1.5]$ is the displacement-decay exponent. Smaller values produce rougher terrain, while larger values make the displacements decay faster and produce smoother terrain



Fractal objects generated by Brownian motion



- The object shape can also be controlled using a “modulation” function $f(x, y)$ (e.g., $f(x, y) = \cos(x) \cdot \sin(y)$), which modulates the random height value by multiplication
- The initialization step may also start from a surface other than a plane (e.g., a polyhedral approximation of a sphere)
- The resulting object can be smoothed, for example, with a Gaussian filter

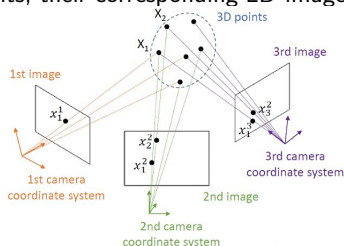


Terrain model generated by the diamond-square method and smoothed with a Gaussian filter



Objects created by reconstructing real-world data

- These objects are created by scanning real-world objects with a laser or camera
- The scanned surface is typically represented as a 3D point cloud, which is then converted into a polygon mesh by triangulation
- The point cloud is acquired with a 3D scanner or by photogrammetry
- Photogrammetry reconstructs a 3D object from images captured from different view-points. To estimate the relative positions and orientations of the cameras, feature point correspondences (points in different images that represent the same point in the scene) and epipolar geometry are used. The 3D coordinates of these points are then determined by triangulation. The camera intrinsic parameters (e.g., focal length, principal point, and pixel aspect ratio) form the calibration matrix K , obtained beforehand from a known calibration target. When a new image is added, *Perspective-n-Point* (PnP) estimates its camera pose from known 3D points, their corresponding 2D image positions, and K



Point cloud obtained using a photogrammetric method [Wessex Archaeology]



Splines

- Used primarily to create smooth curves and surfaces of polynomial degree 2 or 3
- Consist of smoothly joined segments, usually polynomial functions
- Their shape depends on the *control points*, polynomial degree, repeated vertices, knot sequence (nonuniform splines), and weights (rational splines)
- Splines may be interpolating (passing through the control points) or approximating. Common approximating splines, including Bézier curves and B-splines, have the convex-hull property
- A spline curve $\mathcal{K}(t)$ on $[a, b]$ consists of smoothly joined segments. Their number depends on the number of control points and the polynomial degree. In 3D, each segment $P_i(t)$ is a triple $[f_x(t), f_y(t), f_z(t)]$ of polynomial functions:

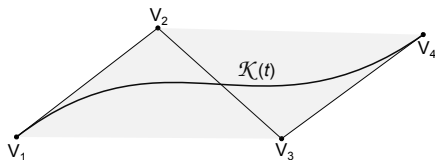
$$\mathcal{K}(t) = \begin{cases} P_0(t), & a = t_0 \leq t < t_1, \\ P_1(t), & t_1 \leq t < t_2, \\ \vdots & \\ P_{k-1}(t), & t_{k-1} \leq t \leq t_k = b \end{cases}$$

- A spline surface $\mathcal{S}(u, v)$ on $[a, b] \times [c, d]$ consists of polynomial segments $P_{ij}(u, v)$

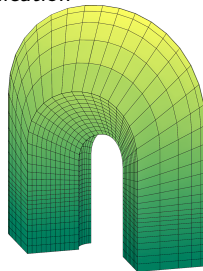


Splines

- Coons patches (interpolating) or NURBS surfaces (approximating) are commonly used for control points arranged in a rectangular grid. For irregularly distributed control points, radial basis functions (RBF) or least-squares approximation can be used
- Splines are used primarily in CAD (Computer-Aided Design) software to model surfaces such as car bodies and aircraft, where both the surface shape and its geometric smoothness are important
- They are now included in almost every 3D modeling application



Spline curve lying in the convex hull of its control points



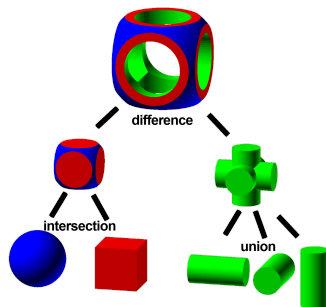
3D object modeled using a NURBS surface

- $\oplus$: Simple implementation and low memory consumption, exact representation of smooth surfaces, simple and intuitive modification of the resulting shape
- $\ominus$: Computing ray-surface intersection points is difficult during rendering



CSG (Constructive Solid Geometry)

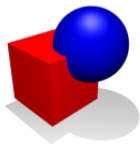
- CSG is one type of volumetric representation for solid objects. It records how an object is constructed from simple geometric primitives instead of explicitly describing its boundary
- **Input:** a set of *primitives* – simple, predefined geometric objects (cone, sphere, cylinder, cube, torus, etc.)
- **Procedure:** combining primitives using a set of regularized binary operations and geometric transformations (translation, scaling, rotation, shearing)
- **Output:** the final object represented, for example, by a binary tree



CSG – a set of regularized binary operations

- A set of simple regularized operations:

- union:
 $O_1 \cup^* O_2$



- intersection:
 $O_1 \cap^* O_2$



- difference:
 $O_1 \setminus^* O_2$



en.wikipedia.org/wiki/Constructive_solid_geometry

- Ordinary set operations may produce lower-dimensional remnants, such as isolated faces, edges, or vertices. These have no volume and therefore do not represent valid CSG solids
- Regularization removes such remnants by taking the interior of the result and then its closure. The result is again a valid volumetric solid or the empty set

- A solid O is regular if

$$O = \text{cl}(\text{int } O)$$

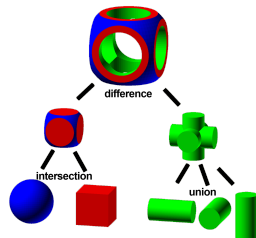
- for each binary operation $\circ$:

$$O_1 \circ^* O_2 := \text{cl}(\text{int}(O_1 \circ O_2))$$



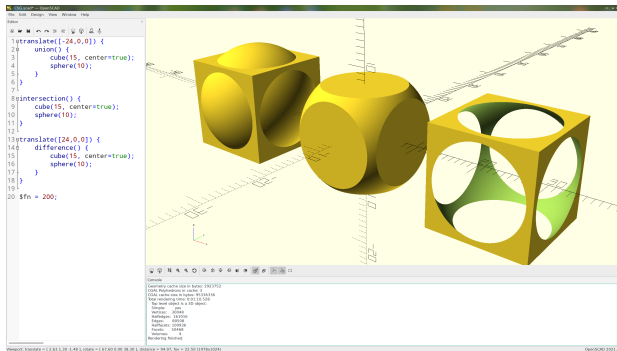
CSG – binary tree

- Geometric primitives are combined using a set of regularized binary operations
- A binary CSG tree represents the final object. Its leaves are geometric primitives, and its internal nodes represent regularized binary operations
- Used mainly to create mechanical components because binary operations imitate the manufacturing process, as well as in CAD systems and procedural modeling
- $\oplus$: Low memory requirements, exact representation of objects built from the available primitives, and a construction history that can support CAM planning for CNC machining
- $\ominus$: Limited set of available input primitives, nonunique representation, and comparison of two objects is not always possible or straightforward
- In CNC (Computer Numerical Control) machining, difference can model material removal, while intersection can model trimming a workpiece to a prescribed shape. Union can model the resulting geometry when two parts are joined



CSG – binary tree

- Such objects can be modeled, for example, using the open source application OpenSCAD



Objects created using union, intersection, and difference in OpenSCAD

CSG – ray intersection (surface rendering)

- To determine the intersection of a ray ℓ with an object created using CSG, the entire tree must be traversed and the same operations used to create the final object must be applied along the ray:

$$\ell \cap (O_1 \circ^* O_2) = (\ell \cap O_1) \circ^* (\ell \cap O_2)$$

- Tree traversal:
 - leaves: intersect the ray with the individual primitives
 - internal nodes: apply binary operations to the ray intersections obtained from the leaves
- Example:

$$\ell \cap (O_1 \cup^* O_2) = (\ell \cap O_1) \cup^* (\ell \cap O_2) =$$

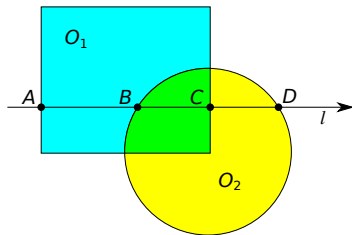
$$= |AC| \cup^* |BD| = |AD|$$

$$\ell \cap (O_1 \cap^* O_2) = (\ell \cap O_1) \cap^* (\ell \cap O_2) =$$

$$= |AC| \cap^* |BD| = |BC|$$

$$\ell \cap (O_1 \setminus^* O_2) = (\ell \cap O_1) \setminus^* (\ell \cap O_2) =$$

$$= |AC| \setminus^* |BD| = |AB|$$



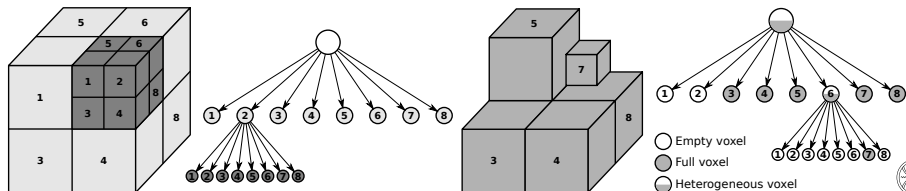
Recursive space subdivision

- A typical volumetric representation of 3D objects, also known as cell decomposition or space subdivision
- Space can be divided into cells of equal size (voxel representation) or different sizes (octrees, kd-trees, BSP trees)
- A 3D object (solid) is then formed by the union of nonempty cells or by their contents
- The goal is to obtain cells that are as “homogeneous” as possible, i.e., each cell contains only one object (or a small number of objects), or only a part of it (or them)
- Voxel representation, a 3D counterpart of two-dimensional pixels, is most commonly used for volumetric data such as medical CT or MRI data
- With a small voxel size (high resolution), voxel representation requires a large amount of memory and does not allow simple modification, such as scaling or rotation, of the represented object



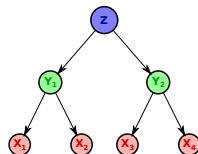
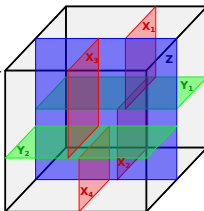
Octrees

- They use variable cell (voxel) sizes. At the center (x_c, y_c, z_c) of the current cell, space is divided into eight equal cells. The splitting planes are $x = x_c$, $y = y_c$, and $z = z_c$, each perpendicular to one coordinate axis
- If a cell contains more than one object, it is subdivided again. This process is repeated until every cell contains one object or is empty
- Subdivision may stop earlier when a minimum cell size is reached or when a cell contains fewer than a predefined maximum number of objects. This reduces memory usage and accelerates the subdivision process
- The subdivision process is recorded in an octree. Its internal nodes represent cells that are subdivided further and have eight children. Its leaves correspond to cells that are empty or contain one object (or the specified maximum number of objects)
- Octrees are useful for visibility computations and accelerate ray-object intersection queries because a leaf contains only one object or part of an object



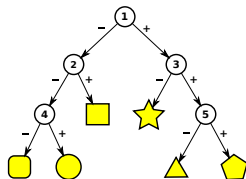
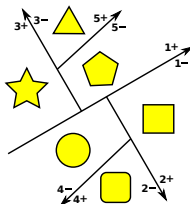
kd-trees

- They generalize octrees (and are a special case of BSP trees). Cell sizes at individual subdivision levels need not be equal because the splitting planes **do not have to** pass through the center of the subdivided cell
- Splitting planes are chosen so that the resulting binary tree is as well balanced as possible
- At each tree level, a cell is divided by only one plane. The splitting axis can be selected independently at each node. The x, y, z axes are often alternated cyclically. The cell is successively split into front and back half-spaces (splitting planes Z), then into upper and lower half-spaces at the next level (splitting planes Y), and finally into right and left half-spaces (splitting planes X)
- The subdivision process is applied recursively to every resulting cell. The termination criterion is the same as for octrees
- Subdivision using a kd -tree is algorithmically more demanding, but it can partition space more efficiently than an octree



BSP (*Binary Space Partitioning*) trees

- The most general method of subdividing space with planes of arbitrary orientation. Each plane divides space into “positive and negative” half-spaces
- If a half-space contains more than one object, it is divided by another plane. The half-space is subdivided recursively until a termination criterion similar to that used for kd-trees is satisfied
- The subdivision process is again stored in a binary tree whose internal nodes represent splitting planes and whose leaves represent convex regions of space containing one object or part of an object
- A suitable choice of splitting planes provides a more efficient space partition and a better-balanced binary tree with fewer levels
- Partitioning using a BSP tree is the most computationally intensive algorithm, but it provides the most flexible partitioning of space
- BSP trees are most commonly used for visibility computations and to accelerate ray-object intersection queries in a scene



Objects defined by implicit functions (F-rep)

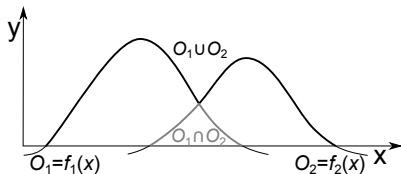
- An implicit function $f = f(x, y, z)$ represents the solid $O_f = \{(x, y, z) \in \mathbb{R}^3 \mid f(x, y, z) \geq 0\}$. Its boundary is defined by $f(x, y, z) = 0$, its interior by $f(x, y, z) > 0$, and its exterior by $f(x, y, z) < 0$. For example, a circle is defined by the equation: $r^2 - x^2 - y^2 \geq 0$
- More complex objects can be created using geometric transformations (e.g., scaling by changing the arguments of the implicit function) and the binary operations $\cup, \cap, \setminus$:

$$f_1 \cup_{\alpha} f_2 := \frac{1}{1 + \alpha} (f_1 + f_2 + \sqrt{f_1^2 + f_2^2 - 2\alpha f_1 f_2}) \quad (\text{union})$$

$$f_1 \cap_{\alpha} f_2 := \frac{1}{1 + \alpha} (f_1 + f_2 - \sqrt{f_1^2 + f_2^2 - 2\alpha f_1 f_2}) \quad (\text{intersection})$$

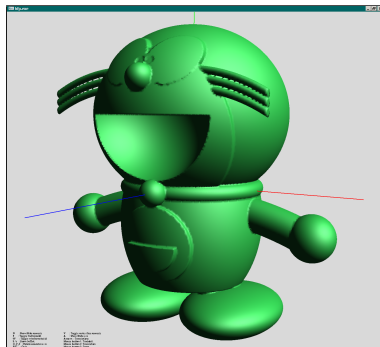
$$f_1 \setminus_{\alpha} f_2 := f_1 \cap_{\alpha} (-f_2) \quad (\text{difference}),$$

where $\alpha = \alpha(f_1, f_2)$ is any continuous function satisfying $-1 < \alpha(f_1, f_2) < 1$. The standard choice is $\alpha = 0$. In the limiting case $\alpha = 1$, $f_1 \cup_1 f_2 = \max(f_1, f_2)$, $f_1 \cap_1 f_2 = \min(f_1, f_2)$, and $f_1 \setminus_1 f_2 = \min(f_1, -f_2)$



Objects defined by implicit functions (F-rep)

- A complex object can be represented by a tree, similarly to CSG
- The result is a single “complex” implicit function to which further operations can be applied, such as *morphing*: $M(\mathbf{x}, t) = (1 - t)f_1(\mathbf{x}) + tf_2(\mathbf{x})$, $t \in [0, 1]$, simulation of uniform growth relative to a point $\mathbf{c}$: $F(\mathbf{x}, t) = f(\mathbf{c} + \frac{\mathbf{x} - \mathbf{c}}{1+t})$, $t \in [0, 1]$, and similar operations



Object created by a single implicit function in HyperFun



Objects defined by implicit functions (F-rep)

- Intersecting a ray with an object often requires solving a complex equation. Therefore, the resulting object is commonly visualized by suitable sampling along the individual coordinate axes and conversion to a polygon mesh
- This representation can produce interesting objects suitable for 3D printing



Metaballs (Blobs)

- They represent objects with a smooth, blended surface defined as an isosurface of a scalar field. Each blob contributes a decreasing field function $f(d)$, where $d = d(x, y, z) = \sqrt{(x - x_c)^2 + (y - y_c)^2 + (z - z_c)^2}$ is the distance from the center $[x_c, y_c, z_c]$
- They can be viewed as particles surrounded by a field whose density (influence) decreases with the distance d from the particle center
- A threshold T must be selected to visualize or define the object surface unambiguously
- The density function $f(d)$ can be chosen freely, but it must be decreasing (e.g., exponentially) and bounded by the interval $[0, 1]$
- When two or more objects are combined, they begin to “blend” at the contact region because the fields surrounding their centers are added
- A typical function defining an object F composed of metaballs can be written as:

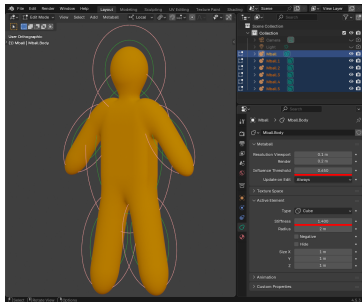
$$F(x, y, z) = \sum_{i \in \text{skeleton}} f_i(d_i) = T, \text{ where } f(d) = -3 \left(\frac{d}{R} \right)^2 + 2 \left(\frac{d}{R} \right)^3 + 1, \text{ for } d \leq R$$

- The constant R bounds the support of the blob field ($f(d) = 0$ for $d > R$), and the skeleton consists of points corresponding to the centers of the individual metaballs



Metaballs (Blobs)

- Other useful density functions $f(d)$ include:
 - $f(d) = e^{-ad^2}$ (Gaussian function in 3D for $a > 0$)
 - $f(d) = \left(1 - \left(\frac{d}{R}\right)^2\right)^2$ for $d \leq R$
 - $f(d) = 1 - \frac{1}{9} \left(22 \left(\frac{d}{R}\right)^2 - 17 \left(\frac{d}{R}\right)^4 + 4 \left(\frac{d}{R}\right)^6\right)$ for $d \leq R$
- Instead of computing ray-metaball intersections analytically, ray marching can sample the implicit field along the ray and search for the threshold level $F(\mathbf{x}) = T$. If a signed distance function or a safe estimate of the distance to the surface is available, the more efficient sphere tracing method can be used



Simple model created with metaballs in Blender