Juraj Onderik | onderik@sccg.sk

Lesson 02

Lesson 03

Scene Representation

Outline of Lessons 02-03

- Representation of curves
- Representation of surfaces
- Representation of volumes

Dimension of Objects

- Object = set of points in n-dimensional space
  - "An object is k-dimensional if there is a continuous one-to-one mapping of the k-dimensional square on this object"
- **0**-dimensional objects = **points**
- **1**-dimensional objects = **curves**
- **2**-dimensional objects = **surfaces**
- **3**-dimensional objects = **solids**

Representation of Curves

What is Curve in CG

- Informal definition
  - Curve is the path of a continuously moving point in the space (2d or 3d) - is the set of all points where the moving point emerge during its motion
- Mathematical descriptions
  - Parametric Curves
  - Implicit Curves
- Application-based classification
  - Interpolation Curves
  - Approximation Curves

Parametric Curves

- Parametric curves in 2D
  - $C(t) = (x(t), y(t))$ where $C: \mathbf{R} \rightarrow \mathbf{R}^2$
  - x and y are any functions $\mathbf{R} \rightarrow \mathbf{R}$
- Parametric curves in 3D
  - $f(t) = (\mathbf{x}(t), \mathbf{y}(t), \mathbf{z}(t))$ where $C: \mathbf{R} \rightarrow \mathbf{R}^3$
  - x, y and z are any functions $\mathbf{R} \rightarrow \mathbf{R}$
- Example: parametric circle
  - $f: (t) \rightarrow (\cos(t), \sin(t)) \quad | \quad t \in [0, 2\pi]$

Implicit Curves

- Implicit curves (only in 2D)
  - Implicit curve C is a set of points (x,y) where a given function c(x,y) is zero
  - $C = \{ (x,y) \mid c(x,y) = 0 \}$ where $c: \mathbf{R}^2 \rightarrow \mathbf{R}$
- Example: implicit circle
  - $C = \{ (x,y) \mid \mathrm{sqrt}(x^2 + y^2) - 1 = 0 \}$

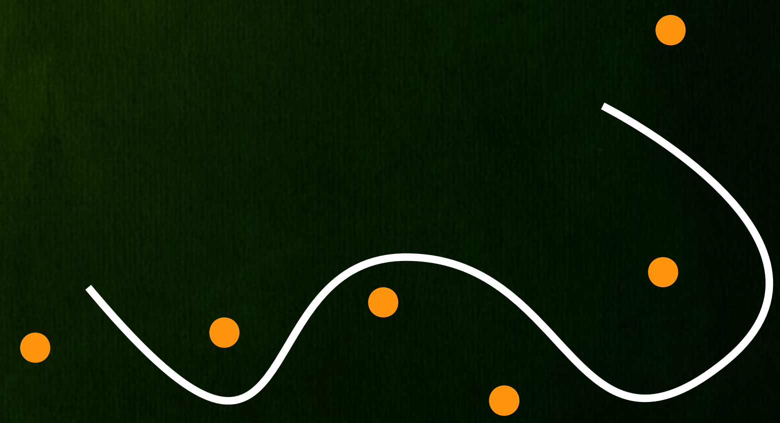
Parametric Polynomial Curves

- Parametric curve $C(t) = (x(t), y(), z(t))$ is polynomial iff functions x,y,z are polynomials
  - $x(t) = x_0 + x_1 t + x_2 t^2 + x_3 t^3 + \ldots + x_{nx} t^{nx} = \sum_{i=0..nx} x_i t^i$
  - $y(t) = y_0 + y_1 t + y_2 t^2 + y_3 t^3 + \ldots + y_{ny} t^{ny} = \sum_{i=0..ny} y_i t^i$
  - $z(t) = z_0 + z_1 t + z_2 t^2 + z_3 t^3 + \ldots + z_{nz} t^{nz} = \sum_{i=0..nz} z_i t^i$
- Curve $C^n(t)$ is n-th degree polynomial if $n_x = n_y = n_z = n$
  - Let $c_k = (x_k, y_k, z_k)$ then
  - $C^n(t) = c_0 + c_1 t + c_2 t^2 + c_3 t^3 + \ldots + c_n t^n = \sum_{i=0..n} c_i t^i$

Application of Curves

- Curves are used in CG mainly for
  - Interpolation of data
  - Approximation of data

Interpolation Curve



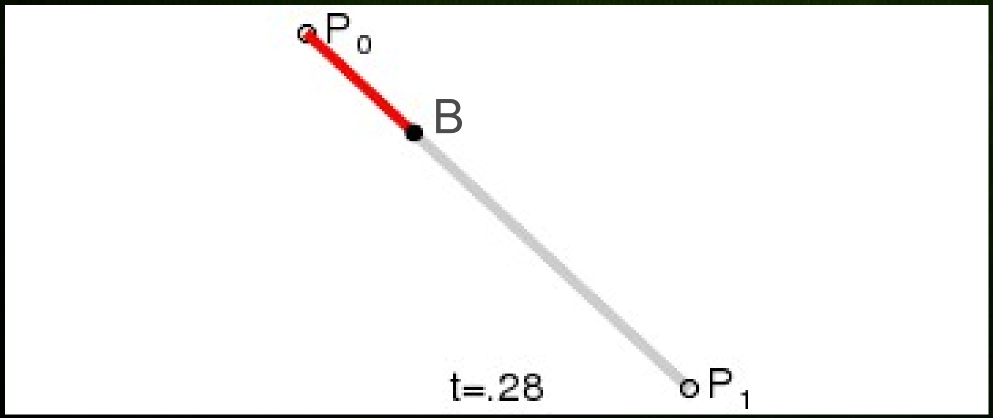
Approximation Curve

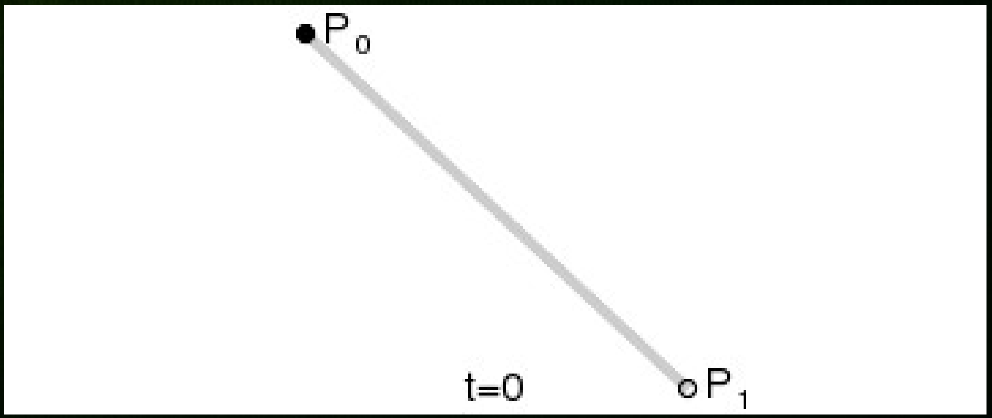
Approximation Curves

- Approximation Curves do not need to interpolate input data points (but can)
- Common approximation curves
  - Bézier Curve
  - B-Spline Curve
  - Catmull-Rom Spline
  - Cardinal Spline...

Linear Bézier Curve

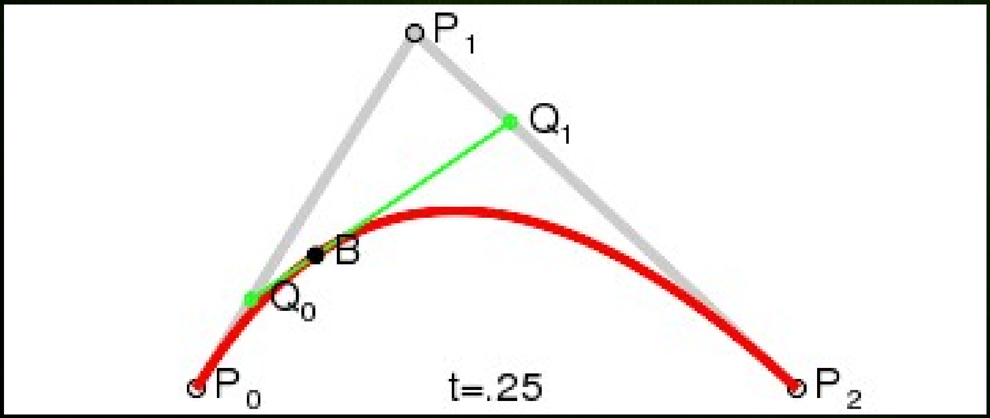
- $B^1(t) = (1-t)P_0 + tP_1$
- Recursive evaluation
  - ➔ $B = (1 - t)P_0 + tP_1$ (linear interpolation)

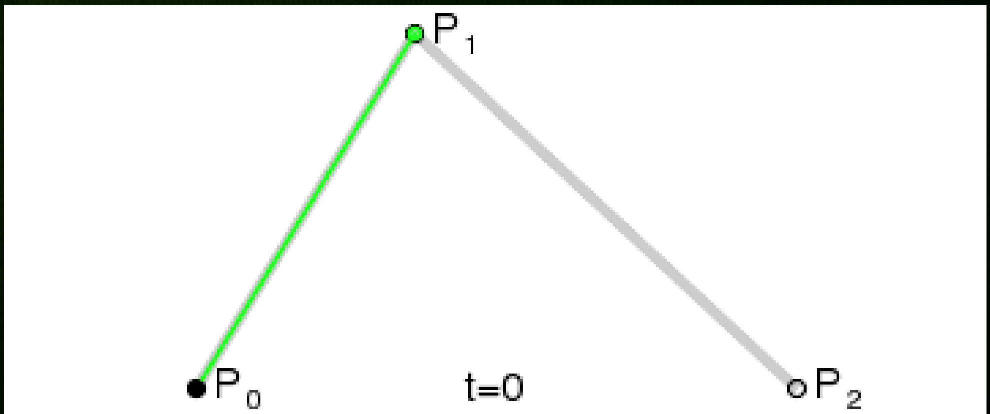




Quadratic Bézier Curve

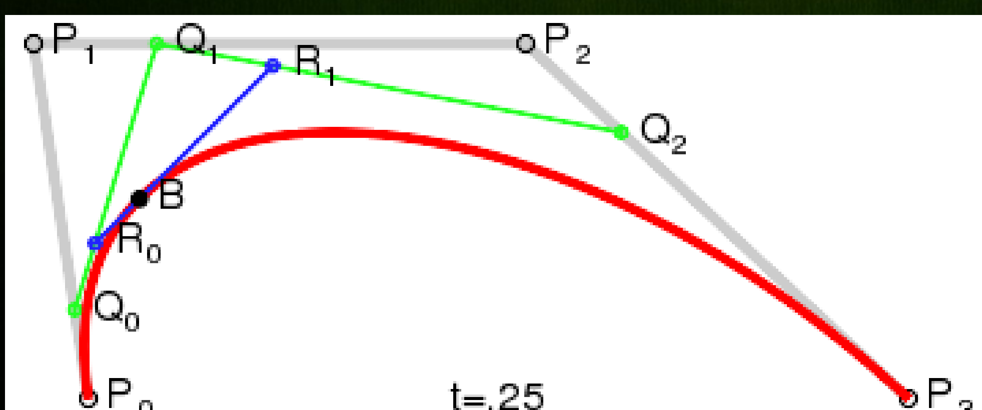
- $B^2(t) = (1-t)^2 P_0 + 2(1-t)tP_1 + t^2 P_2$
- Recursive evaluation:
  - $Q_0 = (1-t)P_0 + tP_1 \quad | \quad Q_1 = (1-t)P_1 + tP_2$
  - $B = (1-t)Q_0 + tQ_1$

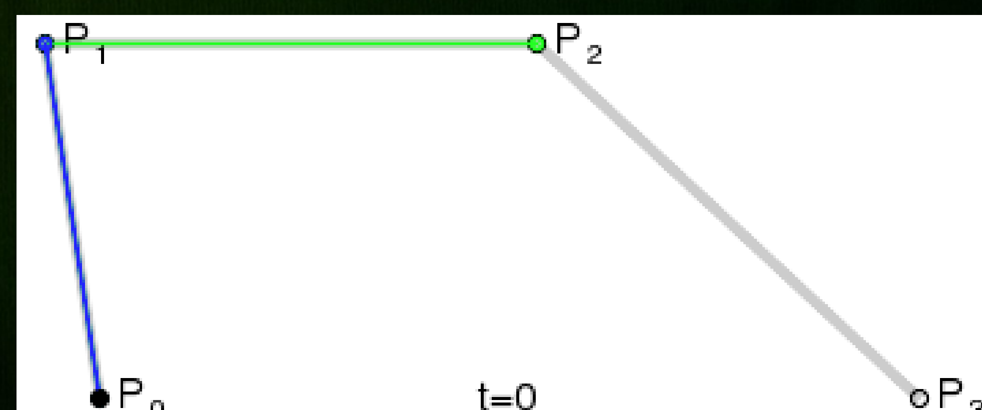




Cubic Bézier Curve

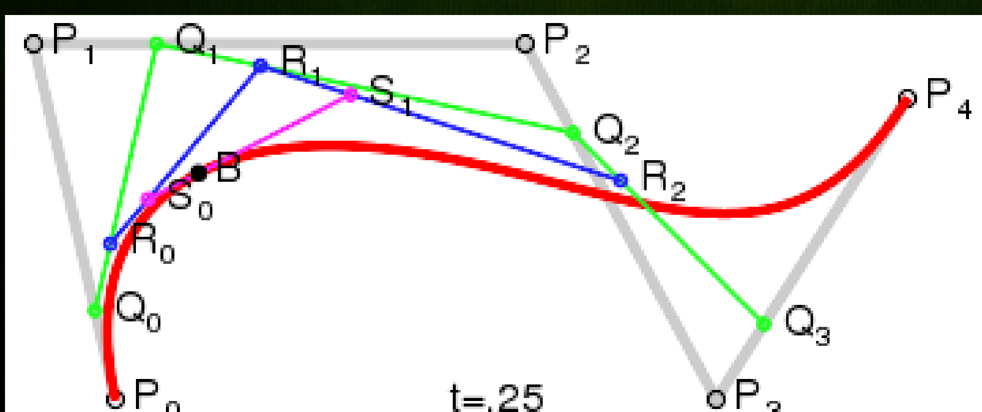
- $B^3(t) = (1-t)^3P_0 + 3(1-t)t^2P_1 + 3(1-t)^2tP_2 + t^3P_3$
- Recursive evaluation:
  - $Q_0 = (1\text{-}t)P_0 + tP_1$ | $Q_1 = (1\text{-}t)P_1 + tP_2$ | $Q_1 = (1\text{-}t)P_1 + tP_2$
  - $R_0 = (1\text{-}t)Q_0 + tQ_1$ | $R_1 = (1\text{-}t)Q_1 + tQ_2$
  - $B = (1\text{-}t)R_0 + tR_1$

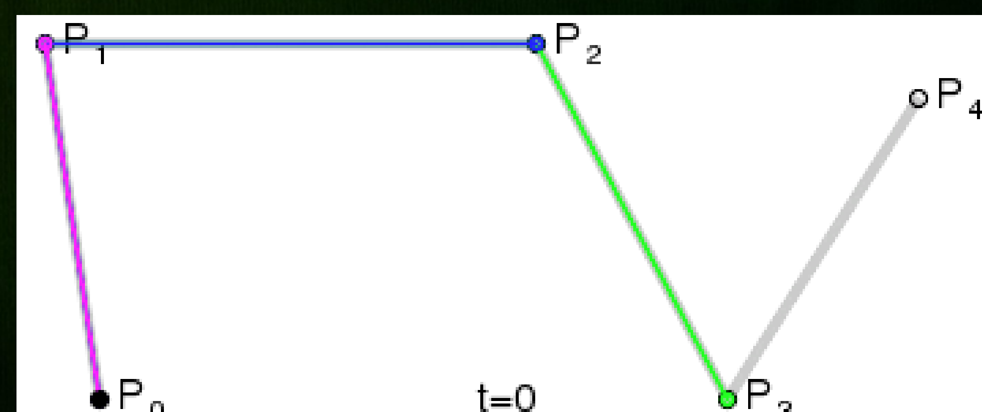




n-th Bézier Curve

- $B^n(t) = \sum_{i=0..n} \binom{n}{i} (1 - t)^{n-i} t^i P_i$
- Recursive evaluation (de Casteljau algorithm)
  - $B^k_i(t) = (1-t) B^{k-1}_i(t) + t B^{k-1}_i(t)$
  - $B^0_i(t) = P_i$
  - $B = B^n_0(t)$



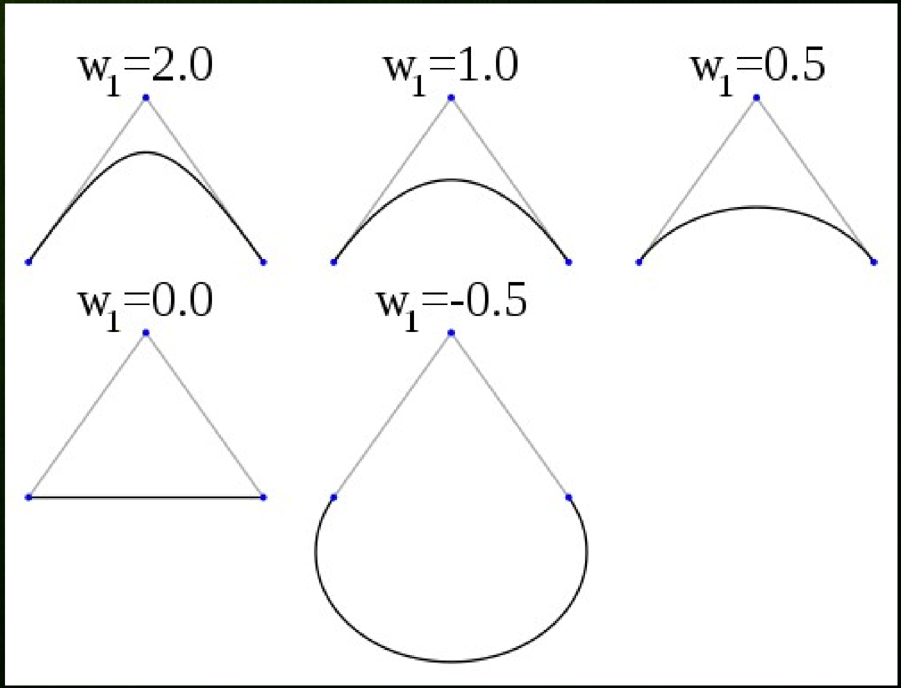


Properties of Bézier Curve

- Interpolation of P_0 and P_n
- Curve is straight line iff all P_i are collinear
- The start (end) of the curve is tangent to the first (last) section of the Bézier polygon
- Always lies in convex hull of Bézier polygon
- Can be split at any point into two Bézier sub-curves
- Each n-th Bézier curve has an equally shaped (n+1)-th Bézier curve (degree elevation)
- Is affine invariant → Affine transformation of curve is equal to curve produced from equally transformed control polygon

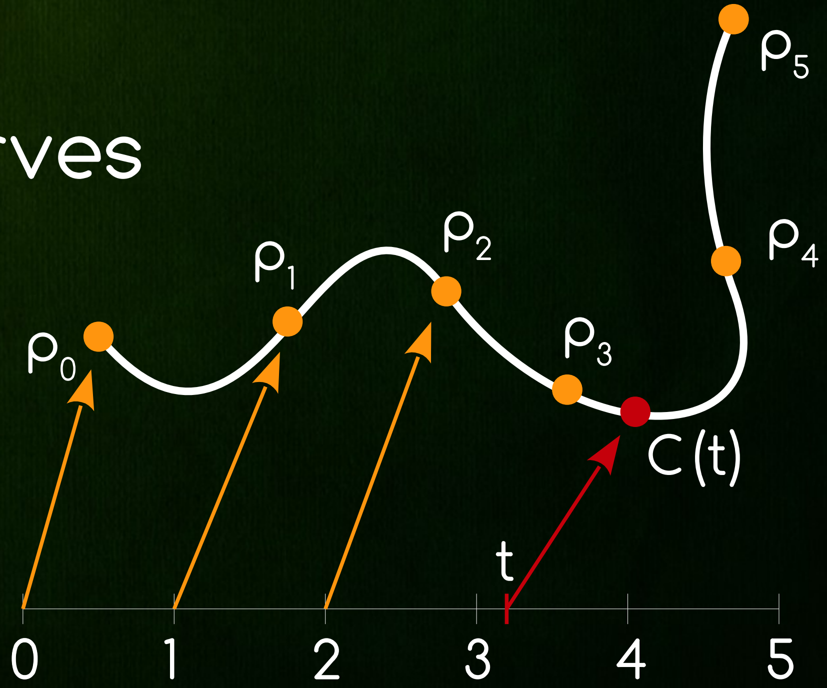
Rational Bézier Curves

- Weighted version of Bézier curve
  - $B^n(t) = (1/W(t)) \sum_{i=0..n} \binom{n}{i} (1 - t)^{n-i} t^i w_i P_i$
  - $W_i(t) = \sum_{i=0..n} \binom{n}{i} (1 - t)^{n-i} t^i w_i$
- Pros
  - Better local control
  - Can express conics
- Cons
  - Need more computation

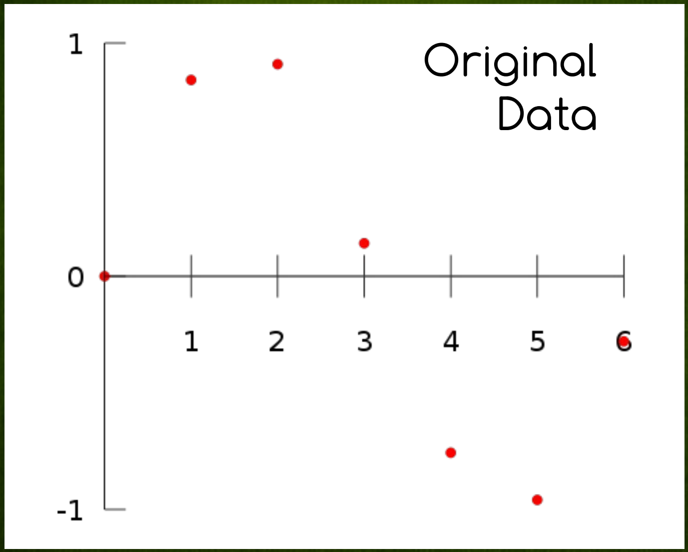


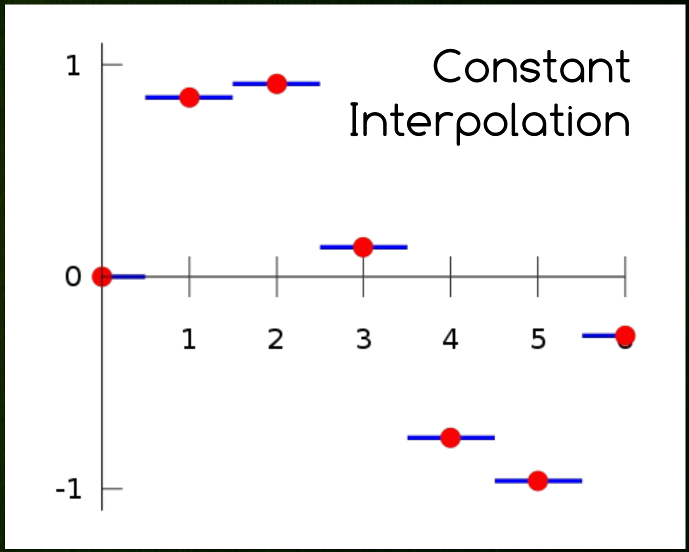
Interpolation Curves

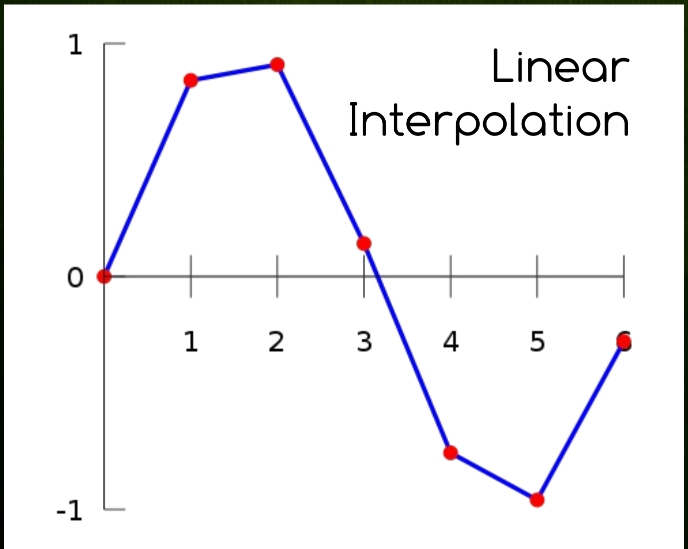
- Given n interpolation points $p_0 \ldots p_{n-1}$ we want to construct an interpolation curve $C(t)$
  - $C(k) = p_k$ where $k = 0 \ldots n - 1$
  - $t \in [0, n-1]$
- Common interpolation curves
  - Lagrange interpolation
  - Piecewise Bezier Curve
  - Piecewise B-Spline Curve
  - Piecewise combinations...

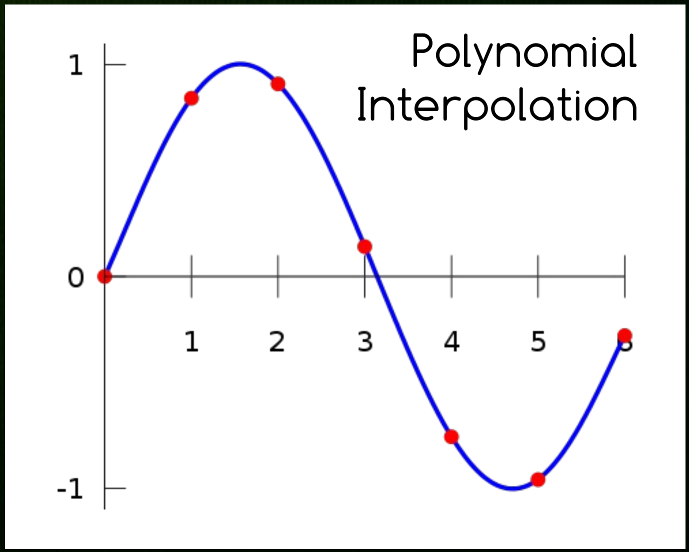


Interpolation Types







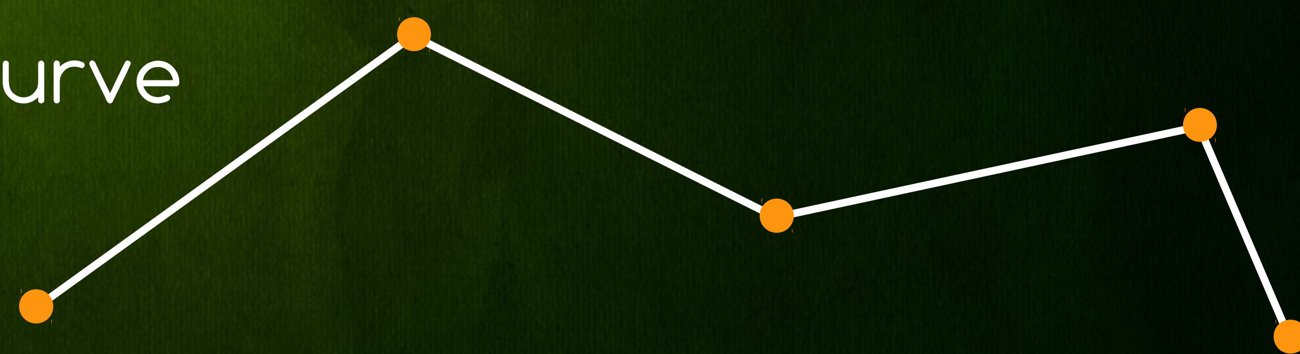


Lagrange Interpolation

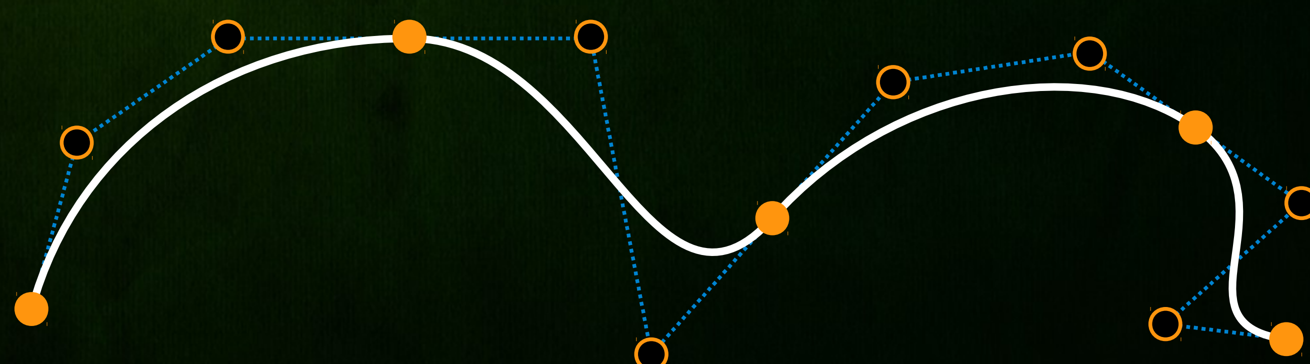
- Given n+1 interpolation points Lagrange interpolation is
  - $L^n(t) = \sum_{k=0..n} l_k(t)\ P_k$
  - $l_k(t) = v_k(t)\ /\ w_k = \prod_{0 \le i \ne k \le n} (t-i)\ /\ (k-i)$
  - $v_k(t) = (t-0)\ldots(t-(k-1))\,(t-(k+1))\ldots(t-n) = \prod_{0 \le i \ne k \le n} (t-i)$
  - $w_k = (k-0)\ldots(k-(k-1))\,(k-(k+1))\ldots(k-n) = \prod_{0 \le i \ne k \le n} (k-i)$
- Pros: Polynomial, easy to implement
- Cons: huge Oscillations, large interpolation error

Piecewise Interpolation Curves

- Known as “Poly-Curves”
  - Each segment between two interpolation points is a given curve
- Linear Poly-curve

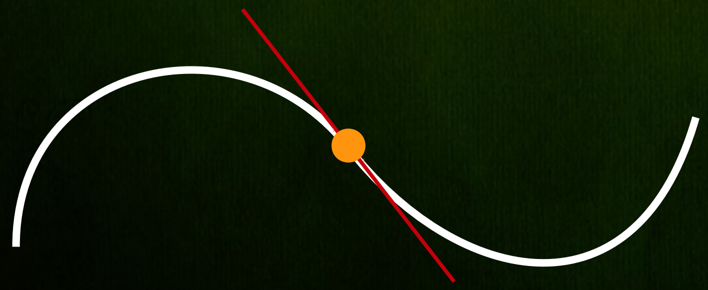


- Cubic Bezier Poly-curve

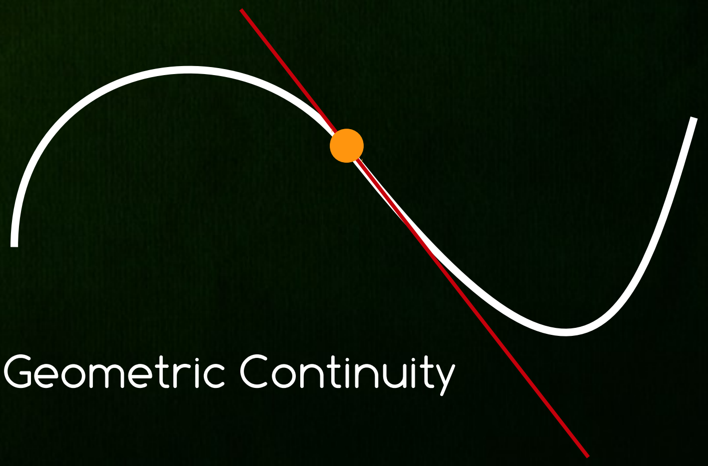


Continuity in Poly-Curves

- Parametric Continuity C^n
  - Segments have equal n-th derivative in interpolations points
  - Tangents have equal **direction** and **length**
- Geometric Continuity G^n
  - Tangents have equal **direction** but not length



Parametric Continuity



Geometric Continuity

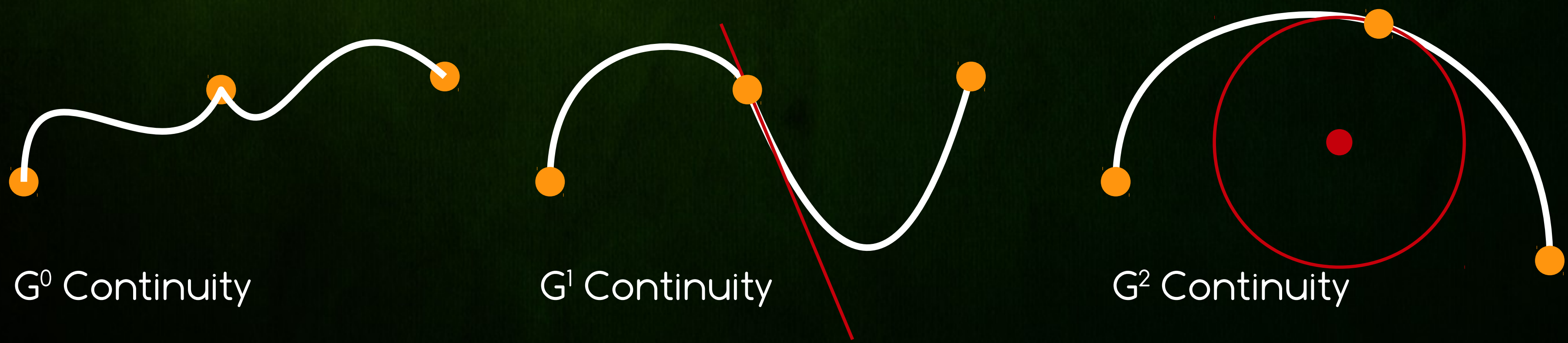
Continuity in Poly-Curves

- Parametric continuity classes
  - C^{-1} = curves include discontinuities
  - C^{0} = curves are joined (continuous)
  - C^{1} = first derivatives are continuous
  - C^{2} = first and second derivatives are continuous
  - C^{n} = first trough n-th derivatives are continuous

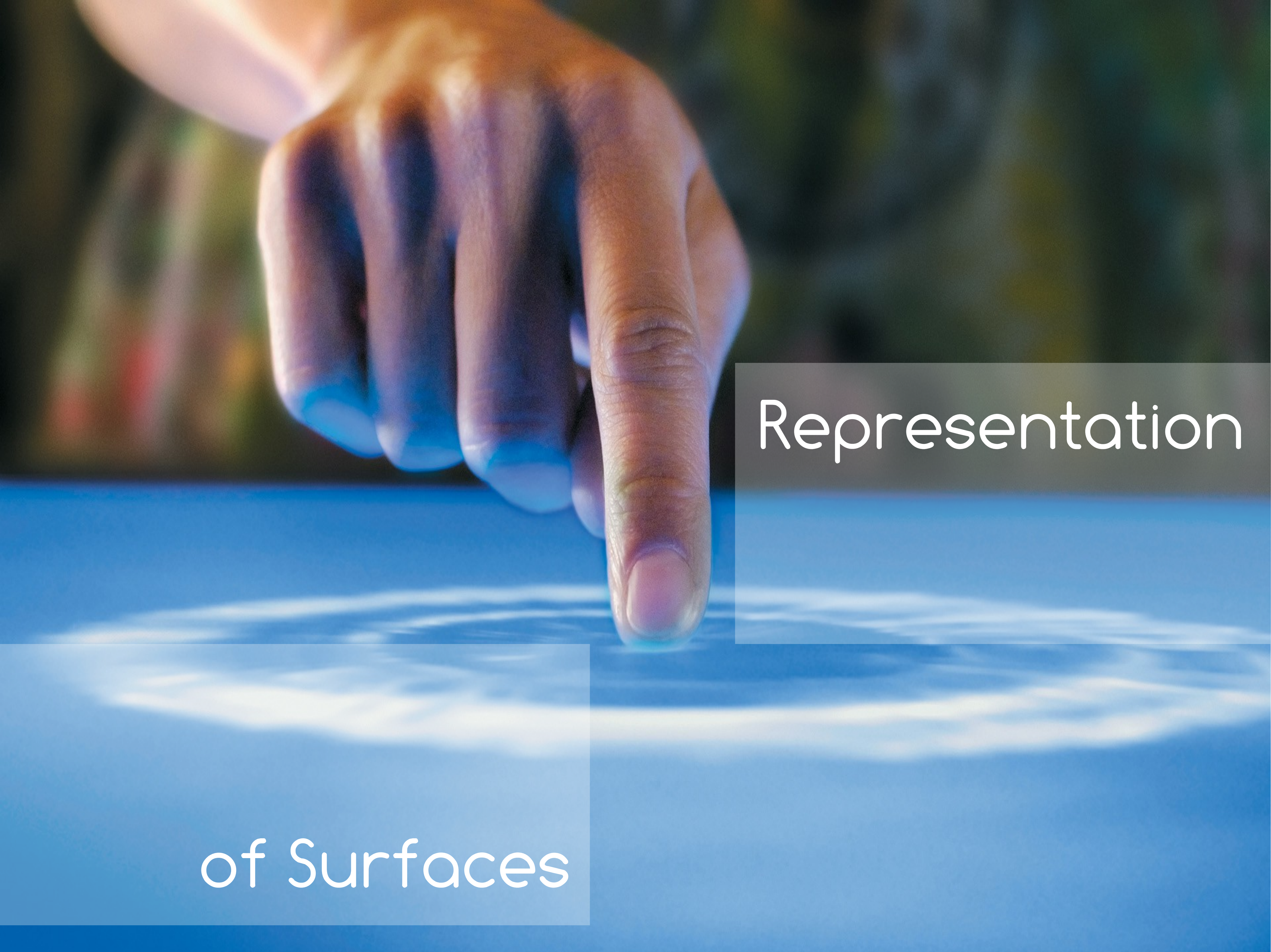
C^{-1} Continuity C^{0} Continuity C^{1} Continuity

Continuity in Poly-Curves

- Geometric Continuity at joint point
  - G^0 = Curves touch at the join point (= C^0)
  - G^1 = Curves share a common tangent direction
  - G^2 = Curves share a common center of curvature
- Curve is G^n continuous if it can be reparametrized to have C^n continuity



G^0 Continuity G^1 Continuity G^2 Continuity


Representation
of Surfaces

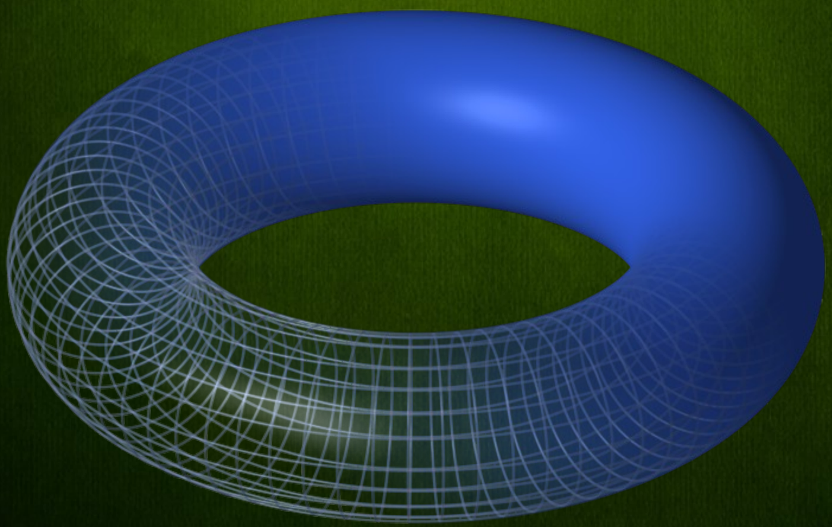
Surface Definition

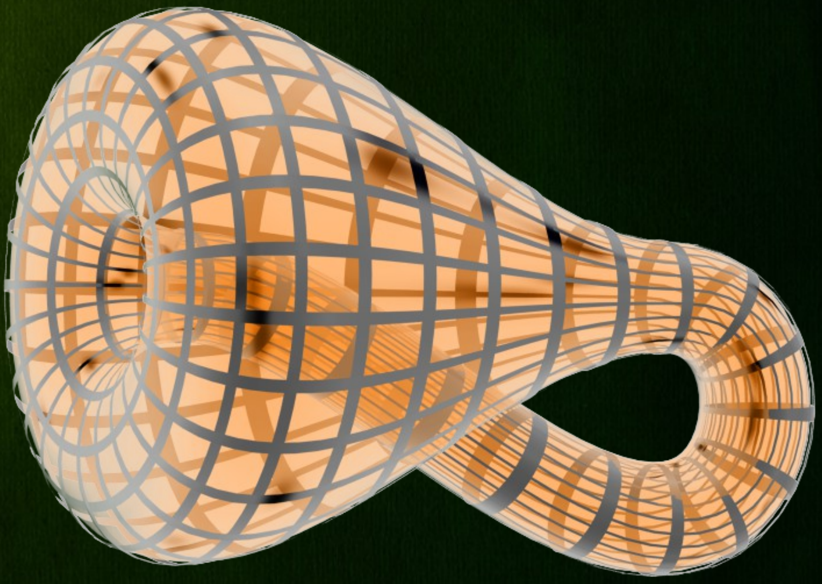
- Formally: "Surface is an orientable continuous 2d manifold embed in $\mathbf{R}^3$"
- Informally: "Surface is the boundary of non-degenerate 3D solid"
- Non-degenerate solid object
  - Each point is the space can be uniquely classified as either interior or exterior w.r.t. given object

Surface Classification

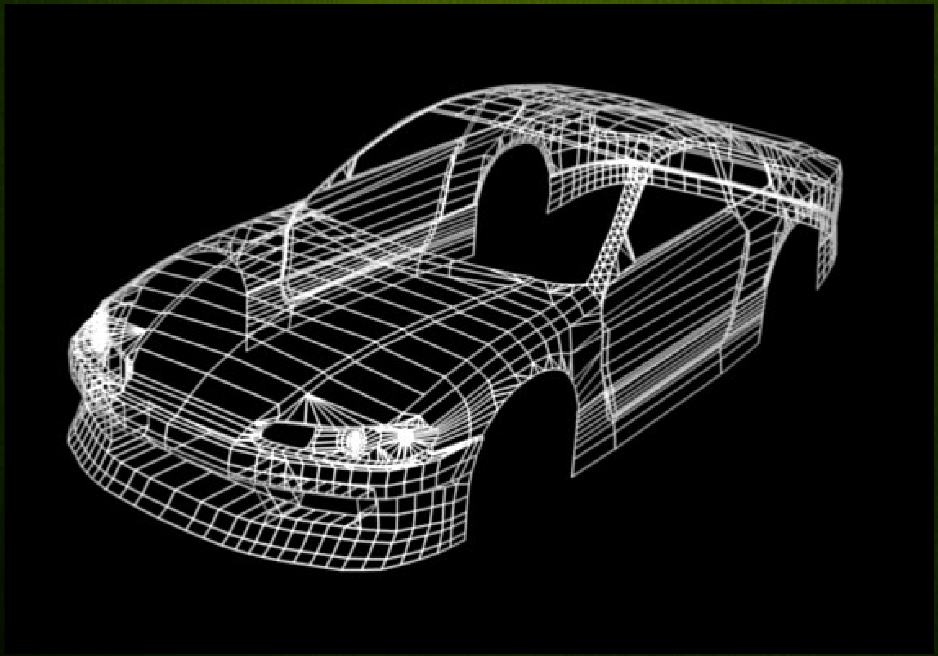
* Orientable / Non-orientable
* Open / Closed (with/without boundaries)
* Manifold / Non-manifold

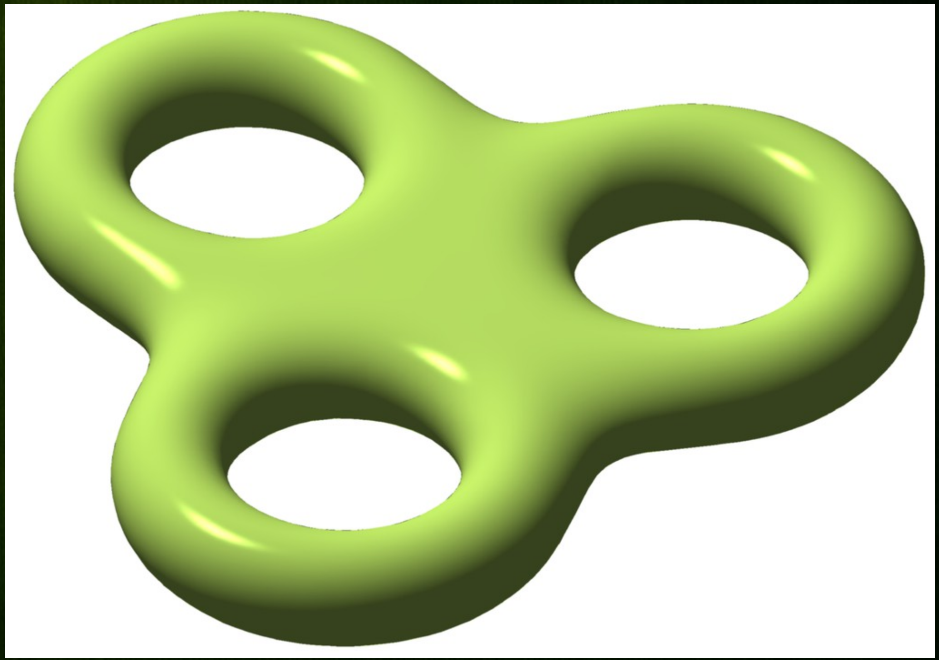
Orientable | Non-orientable



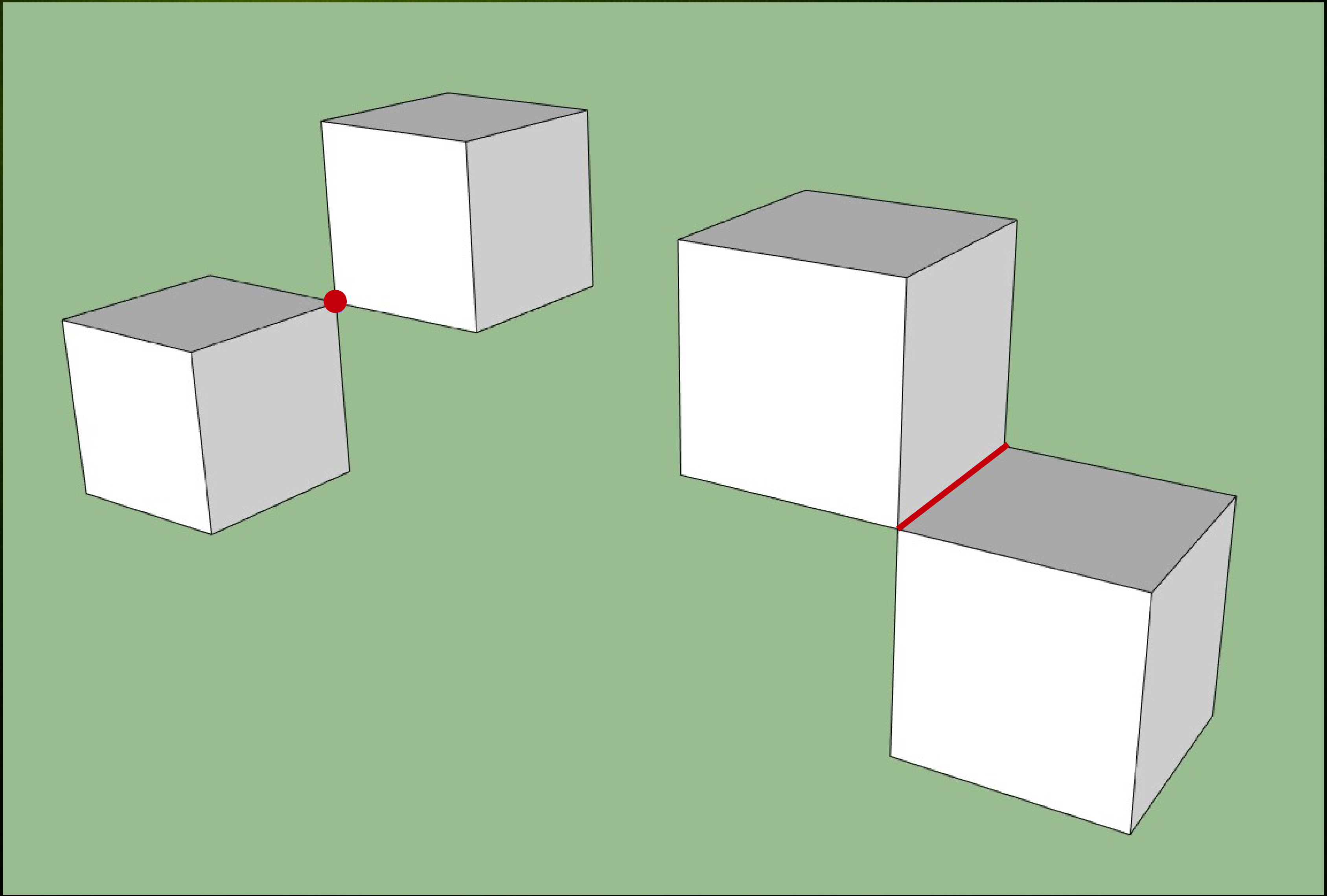


Open | Closed Surface



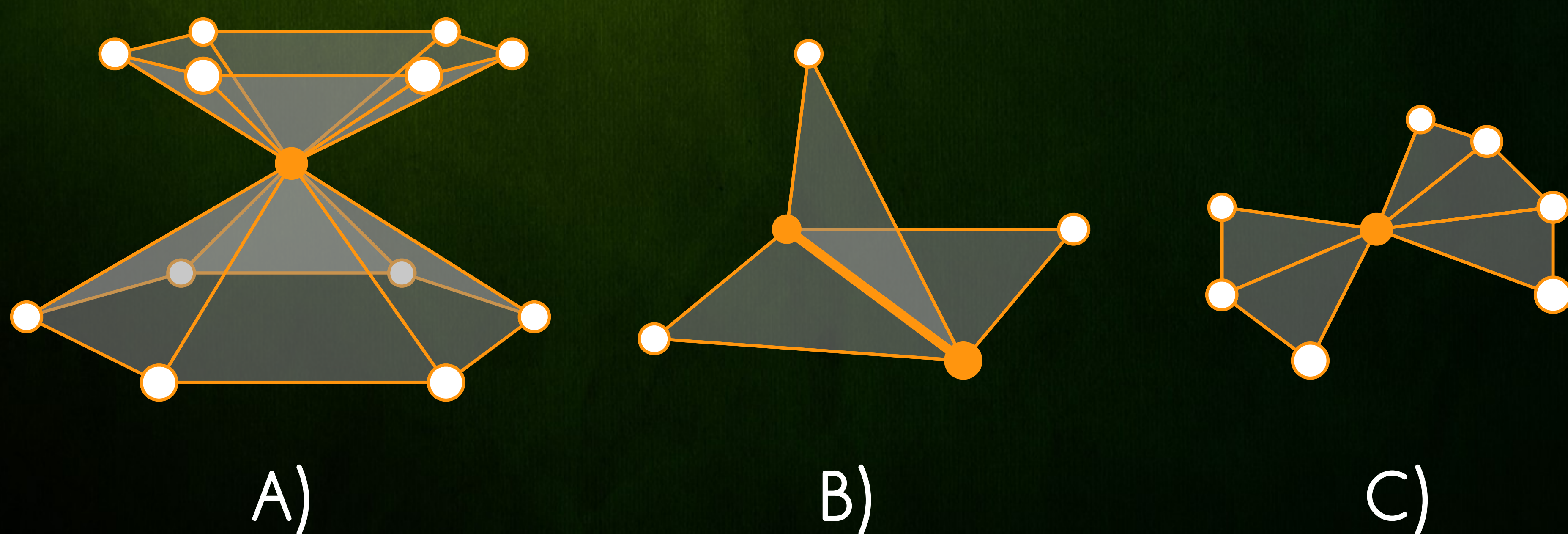


Non-manifold Cases



Non-manifold Cases

- A) Strictly non-manifold vertex
- B) Non-manifold edge
- C) Weak non-manifold vertex

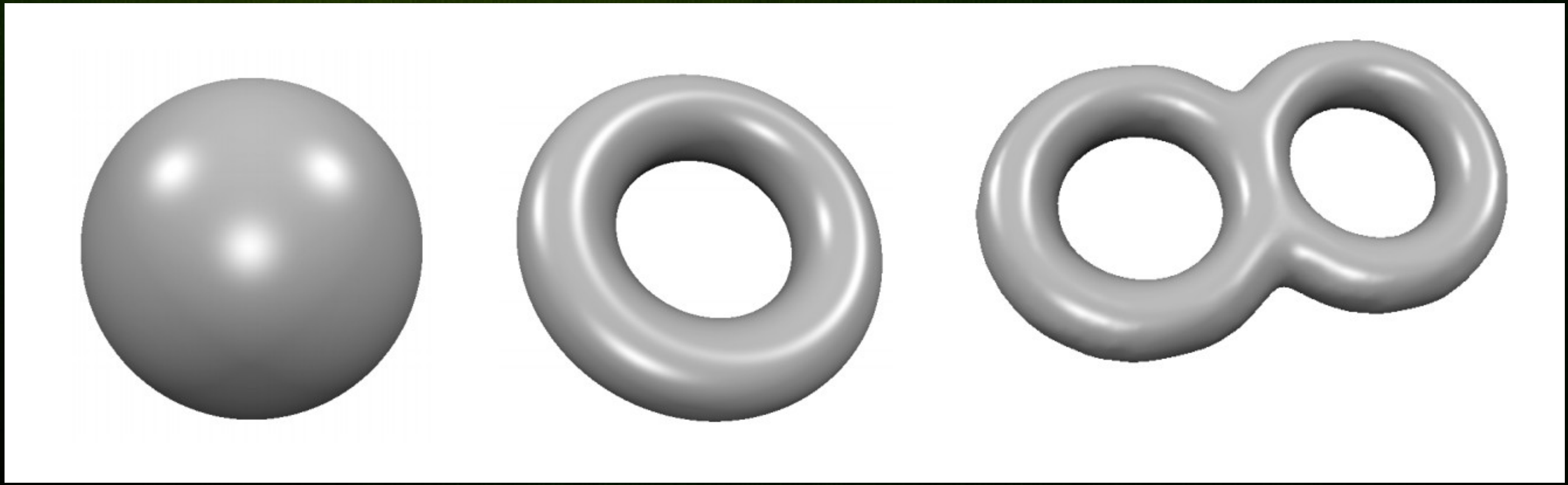


Topological Classification

- Topological equivalence
  - Two surfaces are topological equivalent if we can transform one to each other using only continuous stretching and bending
- Genus of surface
  - The maximum number of cuttings along non-intersecting closed simple curves without rendering the resultant manifold disconnected

Surface Genus

- Genus 0 (Sphere):
  - Surfaces topologically equivalent to sphere
- Genus 1 (Torus): ...
- Genus 2 (Double torus): ...



Operational Classification

- Evaluation
  - The sampling of the surface geometry or of other surface attributes, e.g., the surface normal field.
  - A typical application example is surface rendering
- Modification
  - A surface can be modified either in terms of geometry (surface deformation), or in terms of topology, e.g., when different parts of the surface are to be merged.

Operational Classification

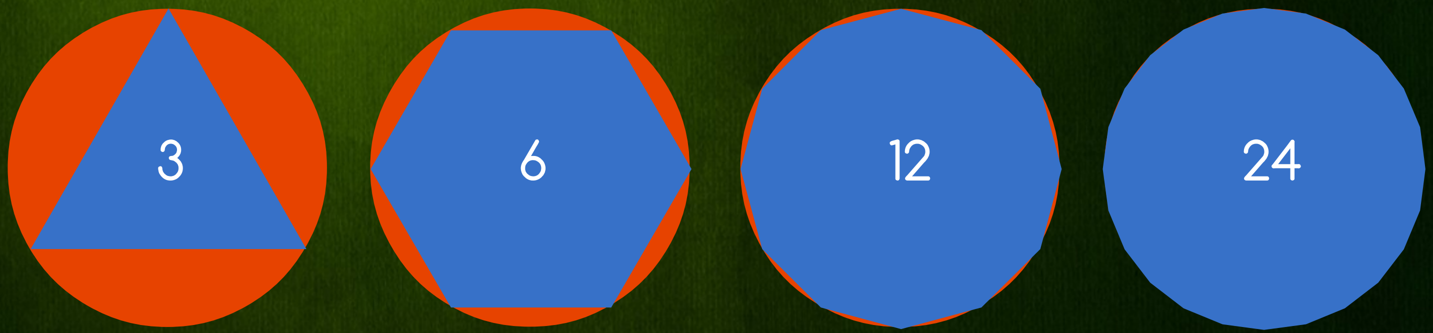
- Query
  - Spatial queries are used to determine whether or not a given point $\mathbf{p} \in \mathbf{R}^3$ is inside or outside of the solid bounded by a surface S
  - This is a key component for solid modeling operations.
  - Another typical query is the computation of a point's distance to a surface.

Parametric vs Implicit Surfaces

- Parametric surfaces
  - [3d] $f: \mathbf{P} \rightarrow \mathbf{C} \;|\; \mathbf{P} \subset \mathbf{R}^2, \mathbf{C} = f(\mathbf{P}) \subset \mathbf{R}^3$
- Implicit surfaces
  - [3d] $f: \mathbf{R}^3 \rightarrow \mathbf{0}$
- Parametric circle
  - $f:(s,t) \rightarrow (\cos(t),\sin(t)) \;|\; f: [0,2\pi]x[0,2\pi] \rightarrow \mathbf{R}^3$
- Implicit sphere
  - $F:(x,y) \rightarrow \mathrm{sqrt}(x^2 + y^2 + z^2) - 1 \;|\; f: \mathbf{R}^3 \rightarrow \mathbf{R}$

Mesh Representation

- Mesh: Piecewise linear approximation with error $O(h^2)$

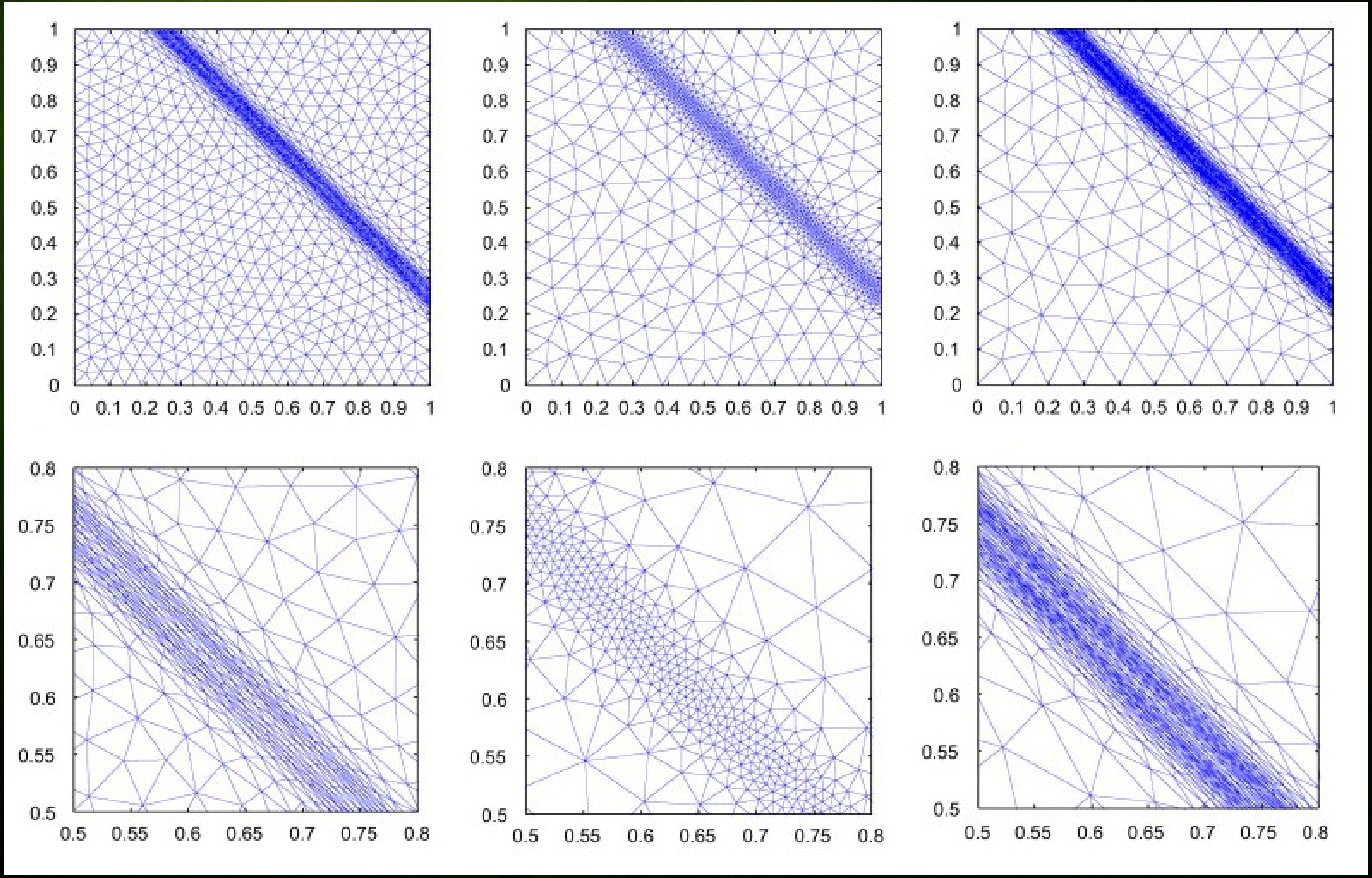


- Mesh elements
  - Face – subset of a 3d plane
  - Edge – Incident points of two (or more) faces
  - Vertex – Incident points of min two edges

Mesh – Local Structure

- Element type
  - Triangular, Quadrilateral meshes…
  - Polygonal (general) meshes
- Element shape
  - Isotropic – locally uniform in all directions
  - Anisotropic – prolong non-uniform elements

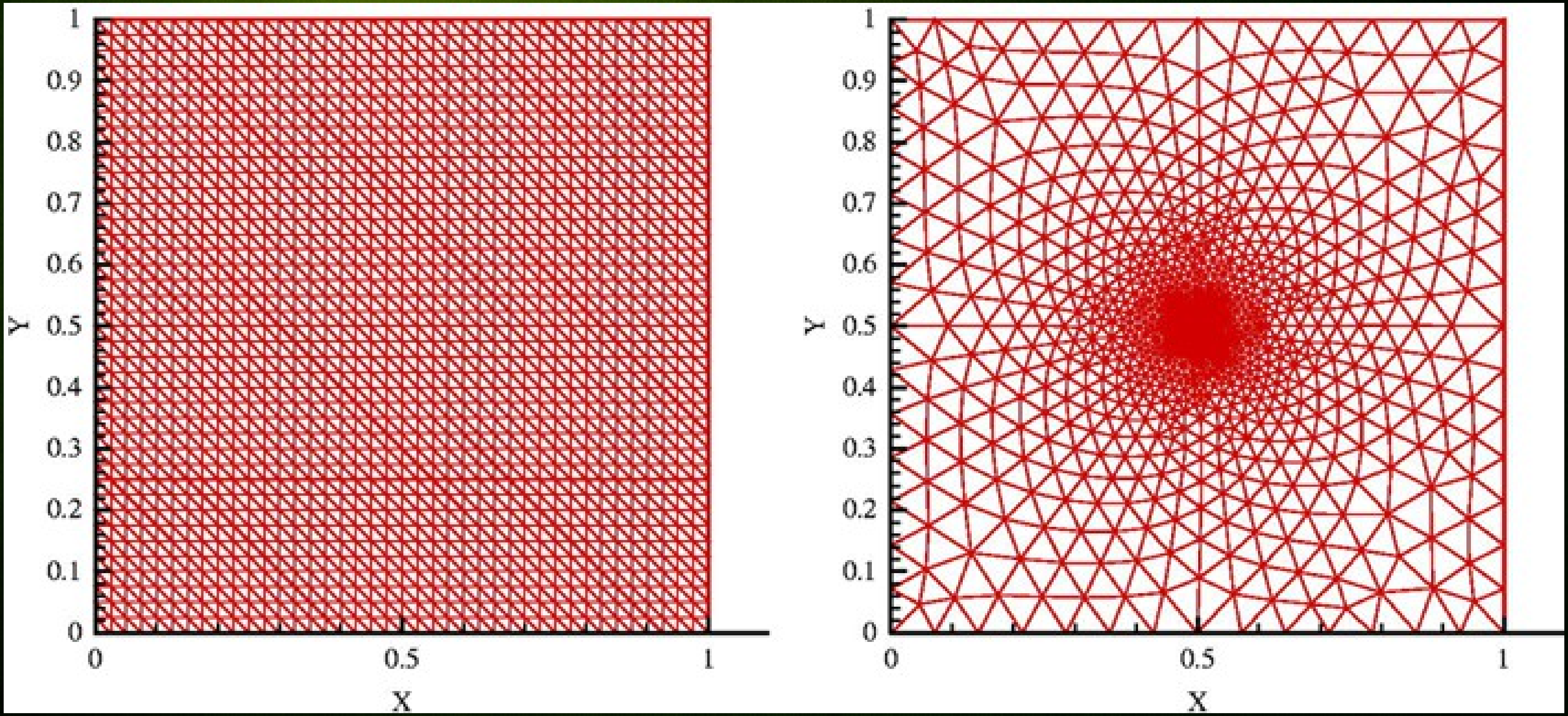
Mesh – Element Shape



Mesh – Local Structure

- Element density
  - Uniform distribution of elements
  - Nonuniform (adaptive) distribution
- Element alignment and orientation
  - Alignment for sharp features of original object
  - Properly represent tangent discontinuities
  - Viable orientation of anisotropic elements

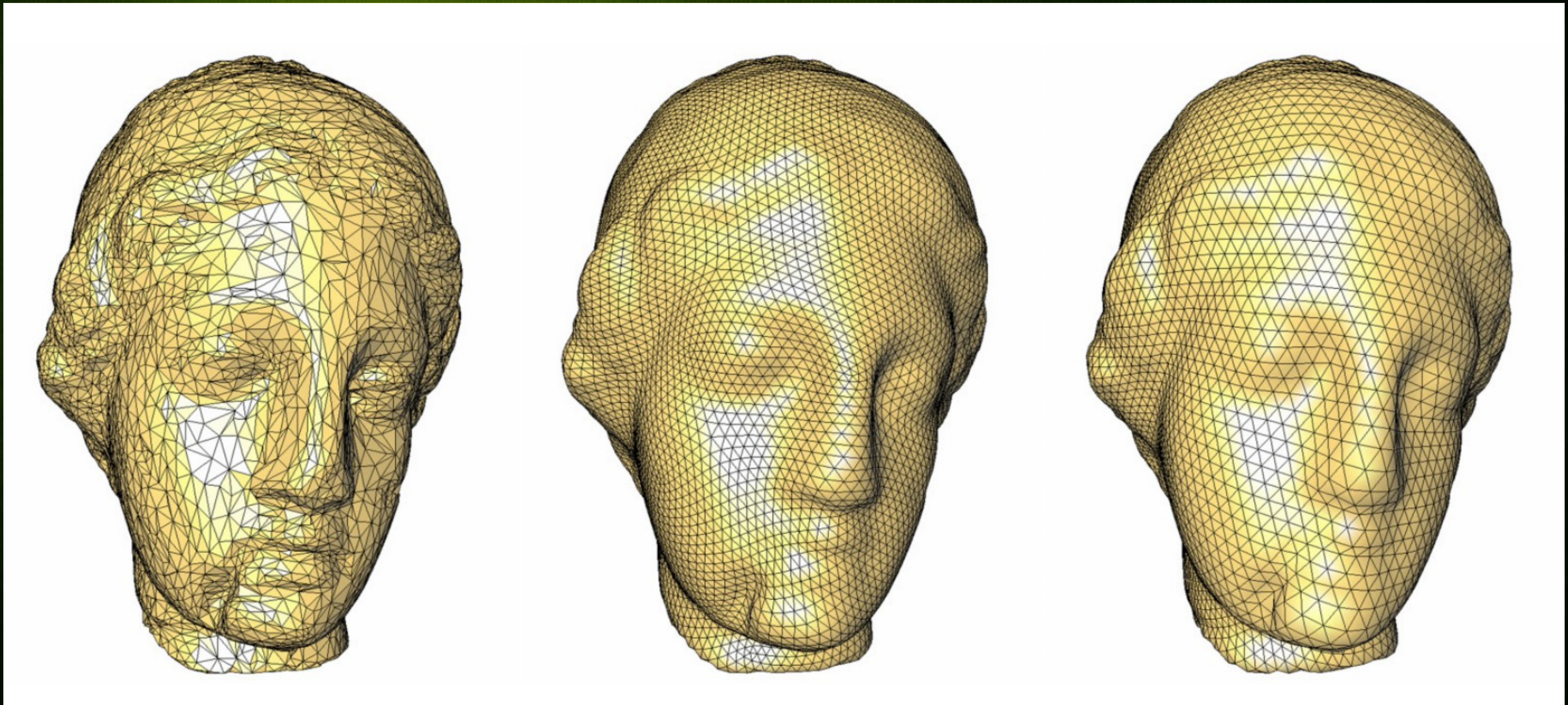
Mesh – Element Density



Mesh – Global Structure

- Topological Complexity
  - 2 - manifolds
  - Complex non-manifold edges, singular vertices
- Regularity
  - Irregular – any number of irregular vertices
  - Semiregular – small number of irregular vertices
  - Highly regular – most vertices are regular
  - Regular – all vertices are regular

Mesh – Regularity



Irregular Mesh | Semi-regular mesh | Regular mesh

Mesh Data Structures

- Face-based data Structures
  - Face Set
  - Indexed Face Set (+ topology data)
- Edge-based data Structures
  - Winged Edge / Quad Edge
  - Half Edge (DCEL)
  - Directed Edge
  - ...

Mesh - Algorithmic Requirements

- What kind of algorithms will be operating on the mesh data structure ?
- Do we need topology data accessible ?
- Do we want to render or edit mesh ? Change topology during editing ?
- What are the memory requirements ? How big will be our mesh ?
- ...

Mesh – Topology Requirements

- Access to individual vertices, edges and faces
  - Enumeration of all elements in unspecified order
- Oriented traversal of the edges of a face
  - Finding previous/next edge in a face
  - Additional access to vertices (for rendering)
- Access to incident faces of an edge
  - Enables access of neighboring (left/right) faces

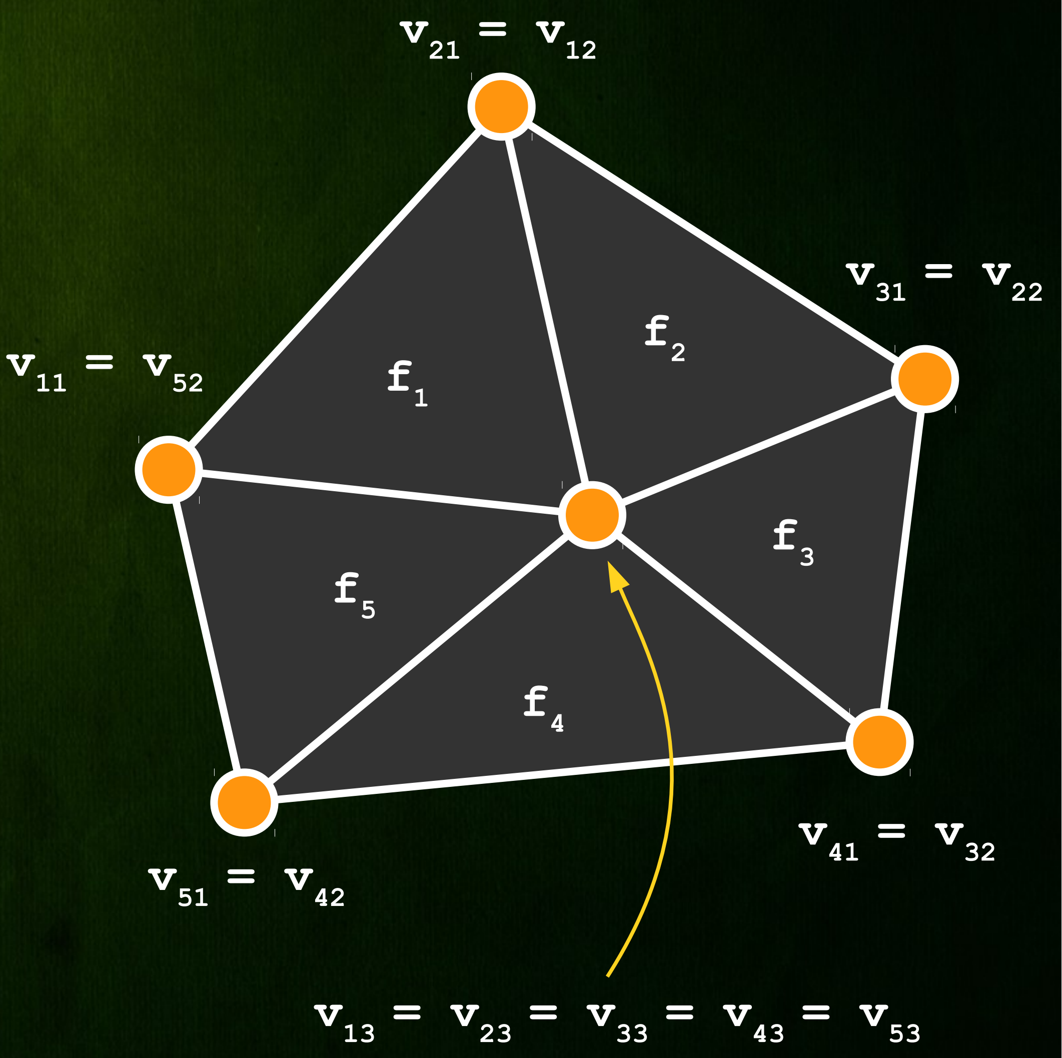
Mesh – Topology Requirements

- Access to vertices of an edge
  - Enables traversal from edge to incident edges
- Access to at least on incident edge/face of vertex
  - For manifold meshes all other elements (edges/faces) in one-ring neighborhood are accessible

Mesh – Face set

Face	
Vertex	$v_1=(x_1,y_1,z_1)$
Vertex	$v_2=(x_2,y_2,z_2)$
Vertex	$v_3=(x_3,y_3,z_3)$

Faces
$f_1=(v_{11},v_{12},v_{13})$
$f_2=(v_{21},v_{22},v_{23})$
...
$f_k=(v_{k1},v_{k2},v_{k3})$
...
$f_F=(v_{F1},v_{F2},v_{F3})$



Mesh – Face Set

- Pros – Suitable for static meshes, rendering
- Cons - No explicit connectivity information. Replicated vertices and associated data
- Storage – 72 bytes per vertex
- Applications - Stereo-lithography
- Performance
  - Rendering – fast
  - One-ring traversal – slow
  - Boundary traversal – slow

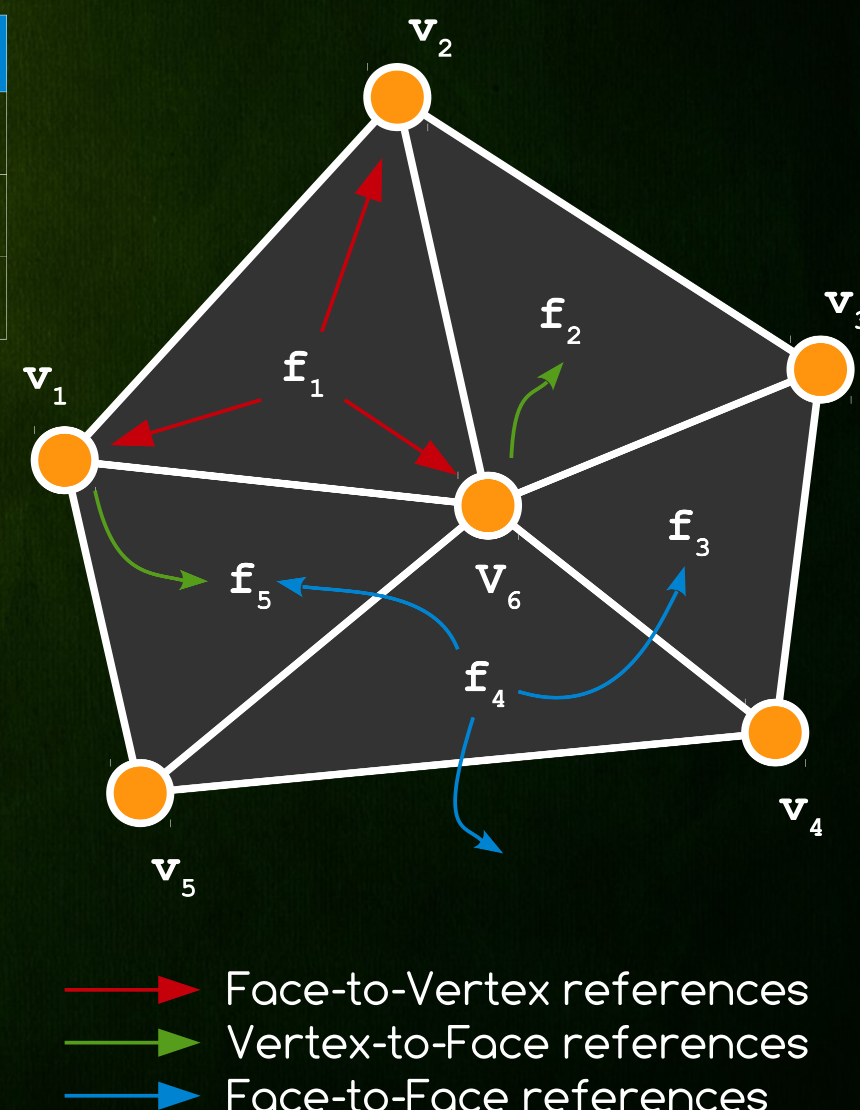
Mesh – Indexed Face Set

Face	
VertexRef	v_1, v_2, v_3
FaceRef	f_1, f_2, f_3
FaceData	data

Vertex	
Point	x, y, z
FaceRef	face
VertexData	data

Faces
$f_1=(i_{11},i_{12},i_{13})$
$f_2=(i_{21},i_{22},i_{23})$
...
$f_k=(i_{k1},i_{k2},i_{k3})$
...
$f_F=(i_{F1},i_{F2},i_{F3})$

Vertices
$v_1=(x_1,y_1,z_1)$
...
$v_V=(x_V,y_V,x_V)$



Mesh – Indexed Face Set

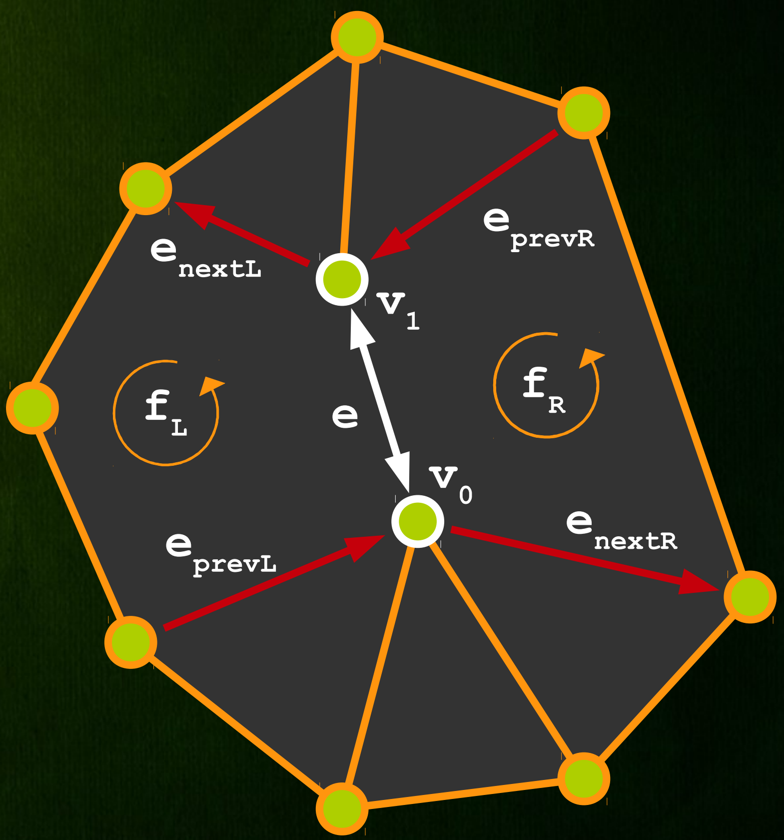
- Pros – Simple and efficient storage. Suitable for static meshes and rendering
- Cons – No explicit connectivity information. Not efficient for most topology algorithms
- Storage – 36 bytes per vertex
- Applications – Rendering (OpenGl, DirectX)
- Performance
  - Rendering – fast
  - One-ring traversal – slow
  - Boundary traversal – slow

Mesh – Winged Edge

Vertex	
Point	position
EdgeRef	edge
VertexData	data

Face	
EdgeRef	edge
FaceData	data

Edge		
VertexRef	v0	v1
FaceRef	fL	fR
EdgeRef	ePrevL	ePrevR
EdgeRef	eNextL	eNextR
EdgeData	data	



Mesh – Winged Edge

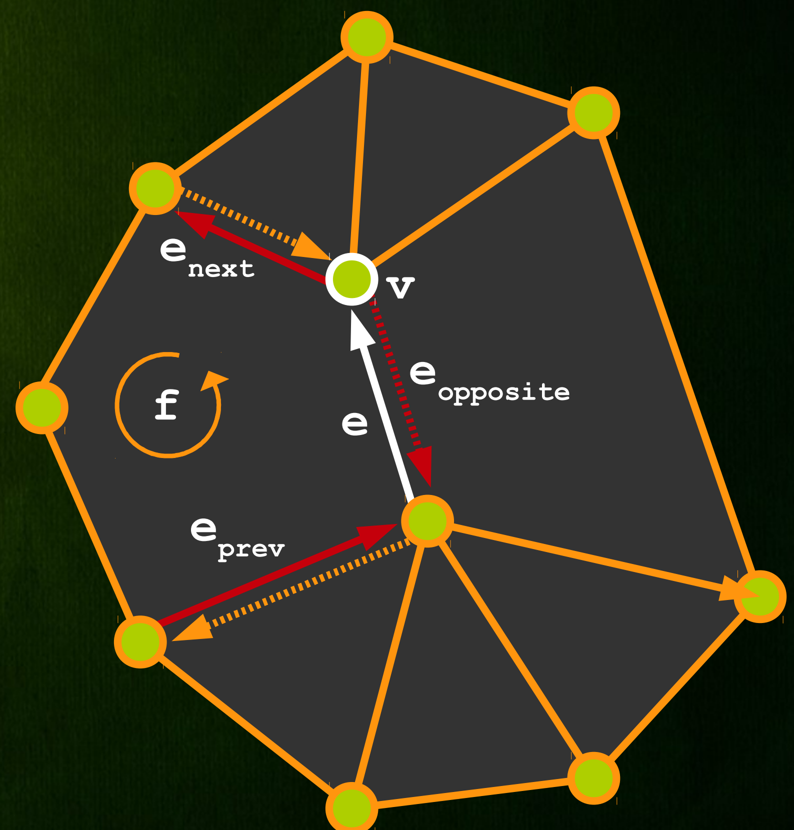
- Pros - Arbitrary polygonal meshes
- Cons - Massive case distinctions for one-ring traversal
- Storage – 120 bytes per vertex
- Applications – Rarely used today
- Performance
  - Rendering – medium
  - One-ring traversal – fast
  - Boundary traversal – medium

Mesh – Half Edge

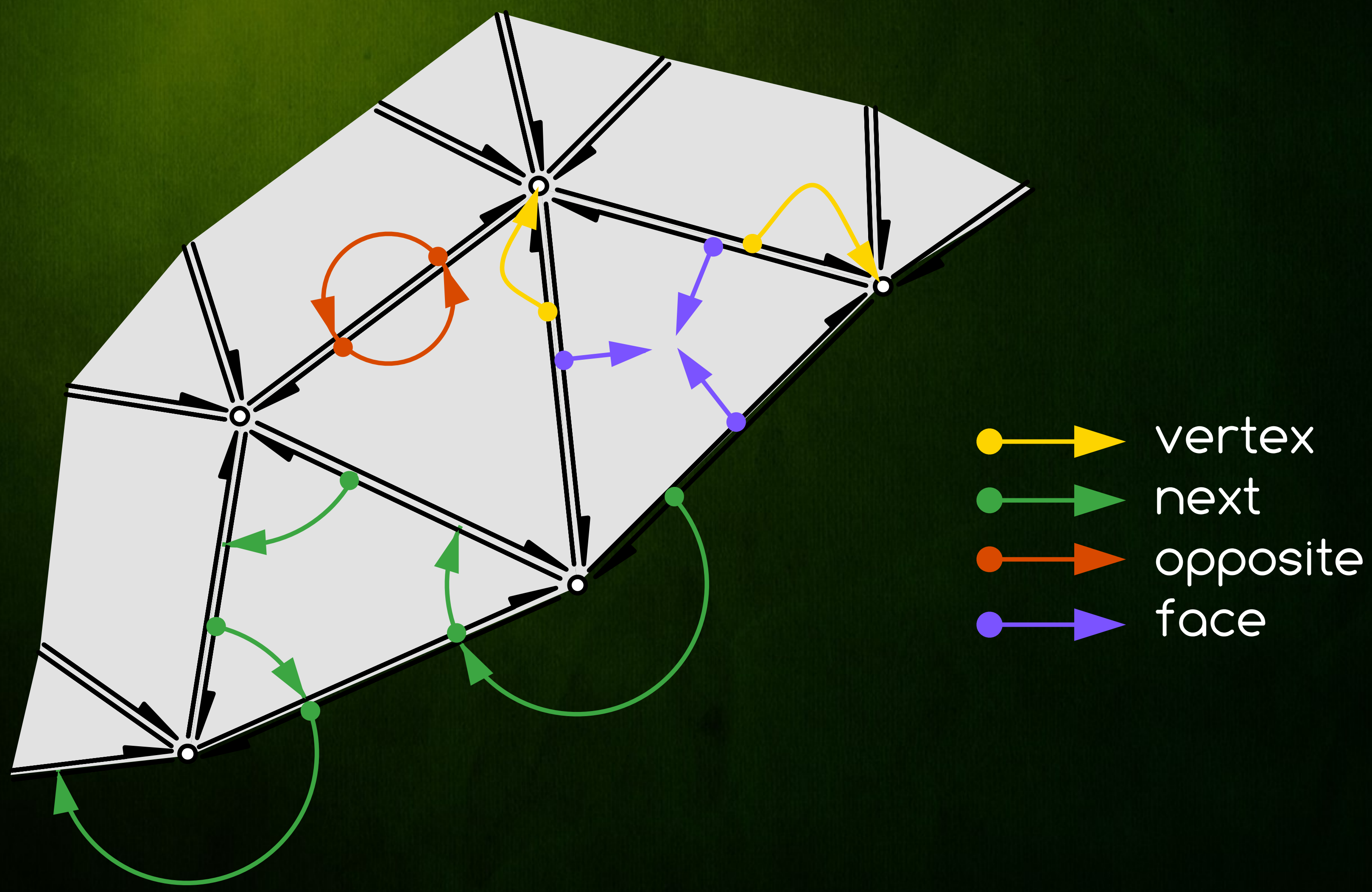
Vertex	
Point	position
HalfedgeRef	edge
VertexData	data

Face	
HalfedgeRef	edge
FaceData	data

Edge	
VertexRef	vertex
FaceRef	face
HalfedgeRef	prev
HalfedgeRef	next
HalfedgeRef	opposite
EdgeData	data



Mesh – Half Edge



Mesh – Half Edge

- Pros – One-ring traversal. Explicit representation of edges
- Cons – Slow rendering
- Storage – 144 bytes per vertex
- Applications - Mostly used for mesh refinement, decimation, smoothing
- Performance
  - Rendering – Medium
  - One-ring traversal – fast
  - Boundary traversal – fast

Mesh – Directed Edge

- Half Edge modification for triangular meshes
  - Store all 3 half-edges of common face next to each other in memory
  - Let f be index of some face. Place its k-th (0,1,2) half-edge on index **hIdx**(f,k) = 3f + k
  - Then h-th half-edge belongs to f-th (= h div 3) face
  - Index of h-th half-edge within its face (= h mod 3)
- We do not need to store face-to-edge and edge-to face references ! They are implicit from face and half-edge storage order

Mesh – Directed Edge

- Pros – Memory efficient, one-ring traversal
- Cons – Only for tri/quad-meshes, no edge info
- Storage – 64b per vertex
- Applications – Mesh refinement, decimation, smoothing of tri-meshes
- Performance – Fast/Medium

Mesh – Performance Comparison

Data Structure	Space / Vertex	Mesh Topology	Rendering	One-Ring Traversal	Boundary Traversal
Face Set	72 bytes	Static, fixed (3,4)	Fast	Slow	Slow
Indexed Face Set	36 bytes	Static, fixed (3,4)	Fast	Slow	Slow
Indexed Face Set + Topology	64 bytes	Usually static	Fast	Fast (if static topology)	Slow
Winged Edge	120 bytes	Any (2 manifolds)	Medium	Slow (case distinctions)	Slow
Quad Edge	144 bytes	Any (2 manifolds)	Medium	Fast	Medium
Half Edge	144 / 96 bytes	Any (2 manifolds)	Medium / slow	Fast	Fast
Directed Edge	64 bytes	Regular Triangular / Quad meshes (2 manifolds)	Medium / slow	Medium	Medium

Mesh – Pros/Cons

Data Structure	Strengths	Weaknesses
Face Set	Static meshes; rendering	No explicit connectivity information; replicated vertices and associated data
Indexed Face Set	simple and efficient storage; static meshes; rendering;	No explicit connectivity information; not efficient for most algorithms
Indexed Face Set + Topology	Access to individual vertices/edges/faces. Oriented traversal; access to incident faces of an edge; access to an edge's two endpoint vertices; one-ring traversal possible	No explicit edge storage (no data attachments); massive case distinctions for one-ring traversal; complex & less efficient for general polygonal faces
Winged Edge	Arbitrary polygonal meshes	Massive case distinctions for one-ring traversal
Quad Edge	One-ring traversal	Slow rendering
Half Edge	One-ring traversal; explicit representation of edges	Slow rendering
Directed Edge	Memory efficiency; One-ring traversal for triangular meshes	Only for pure triangle/quad meshes; no explicit representation of edges

Mesh – Applications

Mesh Data Structure	Common Applications
Face Set	stereo-lithography (STL)
Indexed Face Set	Rendering (OpenGL vertex array, Direct3D), OFF, OBJ, VRML
Indexed Face Set + Topology	2D triangulation data structures of CGAL
Winged Edge	Rarely used today
Quad Edge	Rarely used today
Half Edge	Mostly used for mesh refinement, decimation, smoothing
Directed Edge	Mostly used for mesh refinement, decimation, smoothing of pure triangular meshes

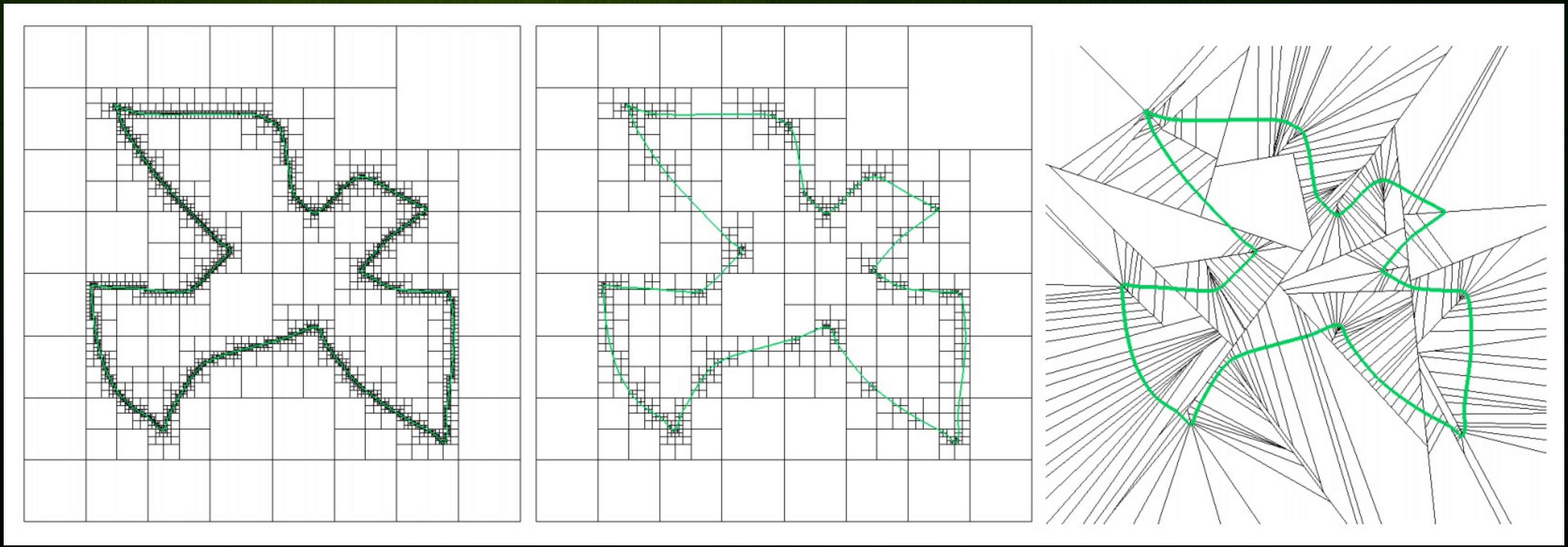
Mesh – Applications

Mesh Data Structure	Common Applications
Face Set	stereo-lithography (STL)
Indexed Face Set	Rendering (OpenGL vertex array, Direct3D), OFF, OBJ, VRML
Indexed Face Set + Topology	2D triangulation data structures of CGAL
Winged Edge	Rarely used today
Quad Edge	Rarely used today
Half Edge	Mostly used for mesh refinement, decimation, smoothing
Directed Edge	Mostly used for mesh refinement, decimation, smoothing of pure triangular meshes


Representation
of Volumes

Volumetric Representations

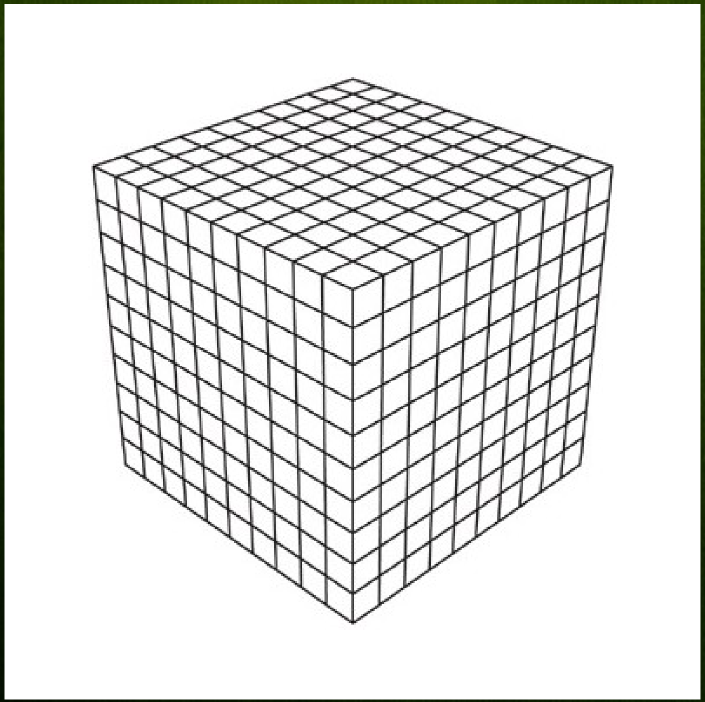
* Spatial subdivision
* Implicit (functional) representations
* Constructive (hierarchical) Geometry

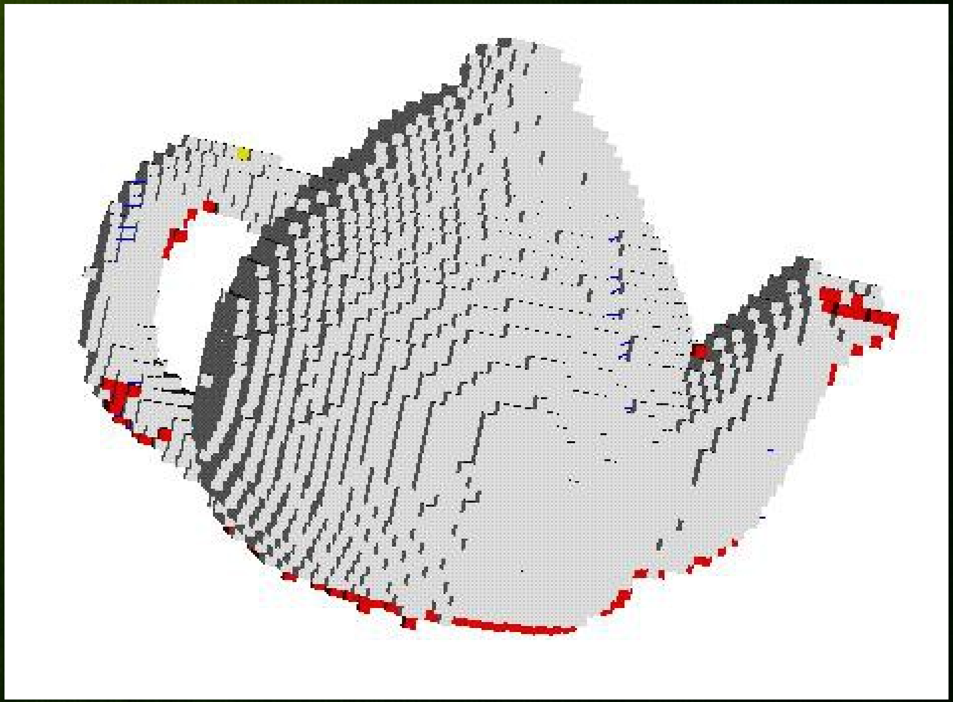


Octree Adaptive Distance Field BSP tree

Uniform Grid

- Trivial 3d regular lattice of N x N x N cells





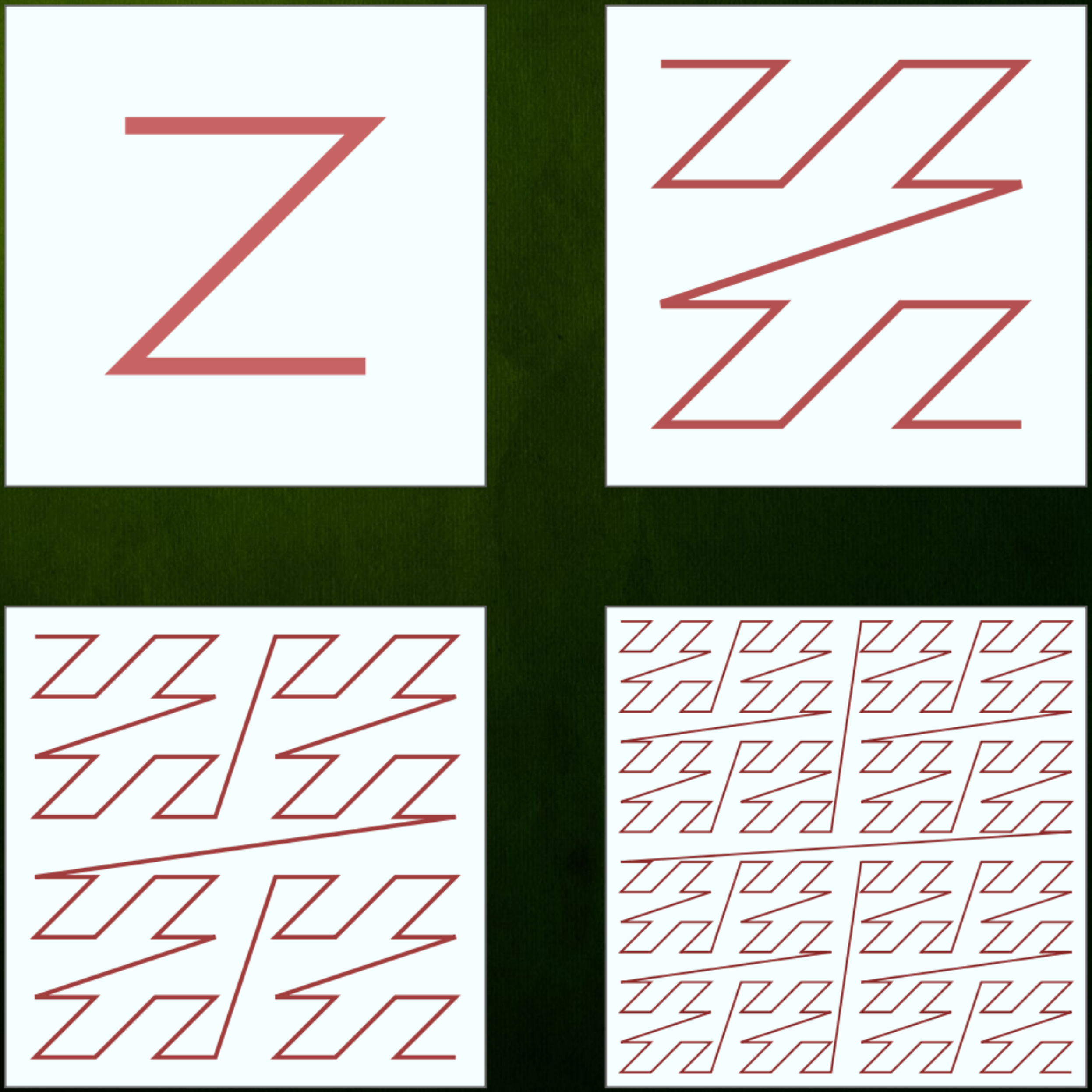
- In each cell we store desired data
  - Color, density, curvature, normal...

Uniform Grid

Construction of Grid

- Find models minimal and maximal coordinates
- Define grid resolution (manual/automatic)
- Choose indexing and create huge linear array in memory
- For each cell (3d loop) sample desired values and store them in cell
- Huge memory footprint !

Uniform Grid – Z-Index

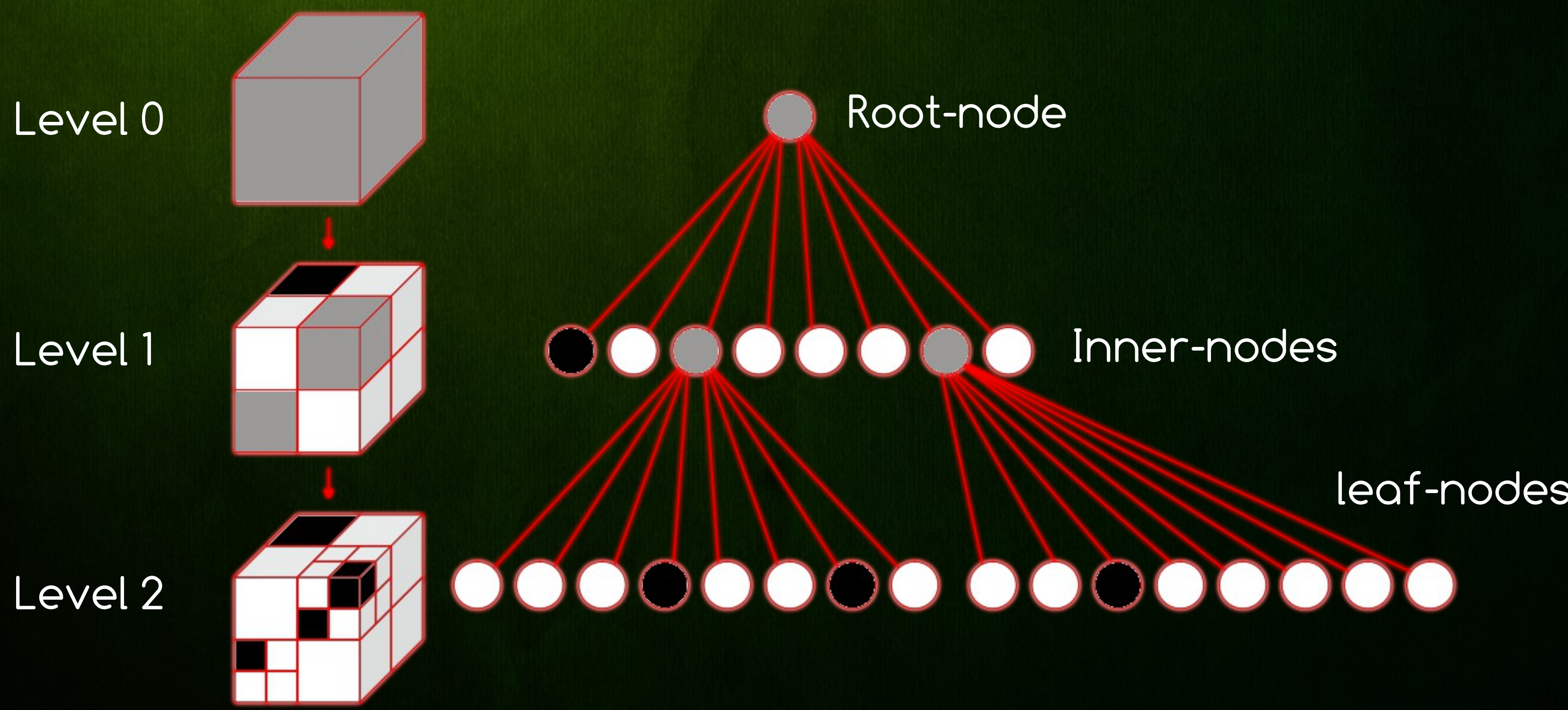


Uniform Grid - Summary

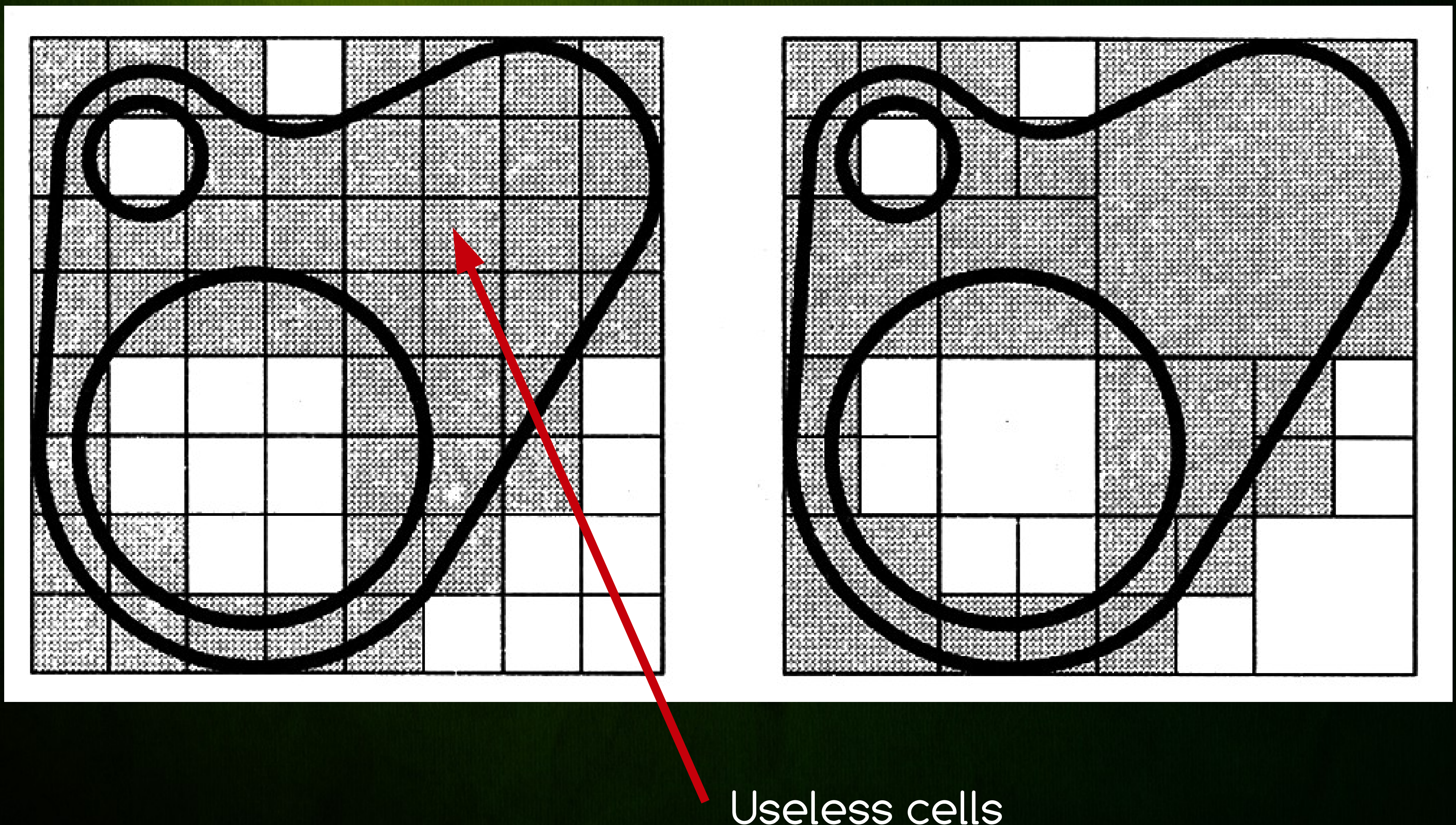
- Pros
  - Trivial data structure
  - Algorithms can be naturally parallelized
  - Natural acquisition for some applications
  - Trivial Boolean operations
- Cons
  - Huge memory requirements (storing empty cells)
  - Large 3d loops make algorithms too slow
- Applications
  - Medical Imaging, Many GPGPU applications

Octree

- Octree is an adaptive hierarchy of cells created only within important (non-empty) data regions. Each non-leaf cell is subdivided exactly into 8 half-size sub-cells



Grid vs Octree

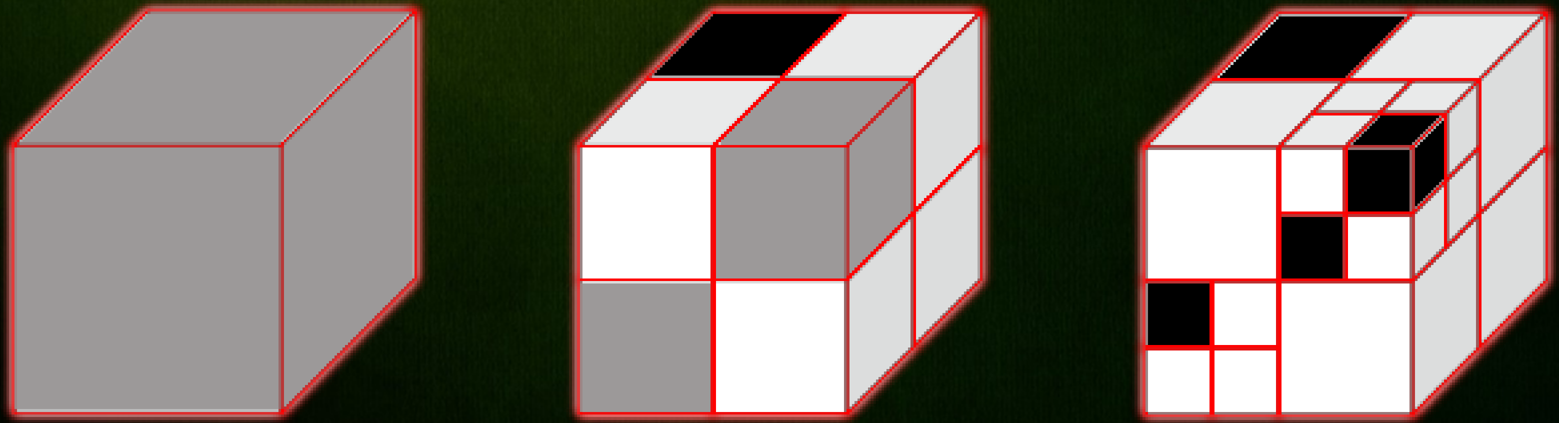


Octree Data Structure

- Node
  - NodeType type
  - NodeRef subNodes[8];
- NodeType
  - Empty – all 8 sub-cells are empty
  - Mixed – there is at least on non-empty sub-cell
  - Full – all 8 sub-cells are full

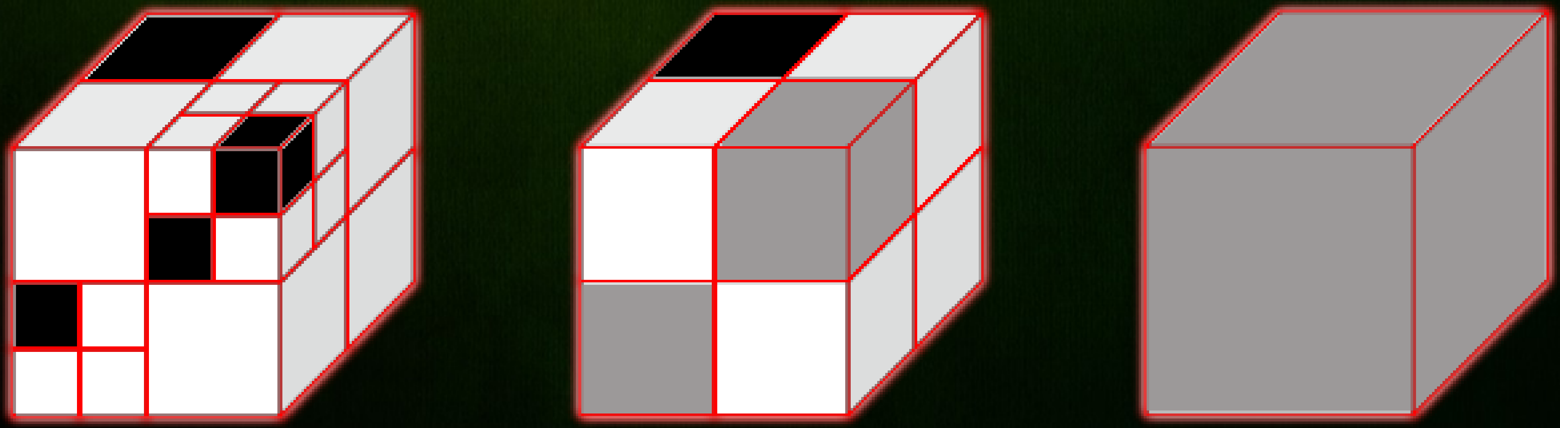
Octree Construction

- Top-Down (slitting) scheme
  - Fit whole data (geometry) into one bounding cell
  - If it is mixed split it into 8 sub-cells
  - Repeat this with each of 8 sub-cells until there is nothing more to split (all are small / empty / full

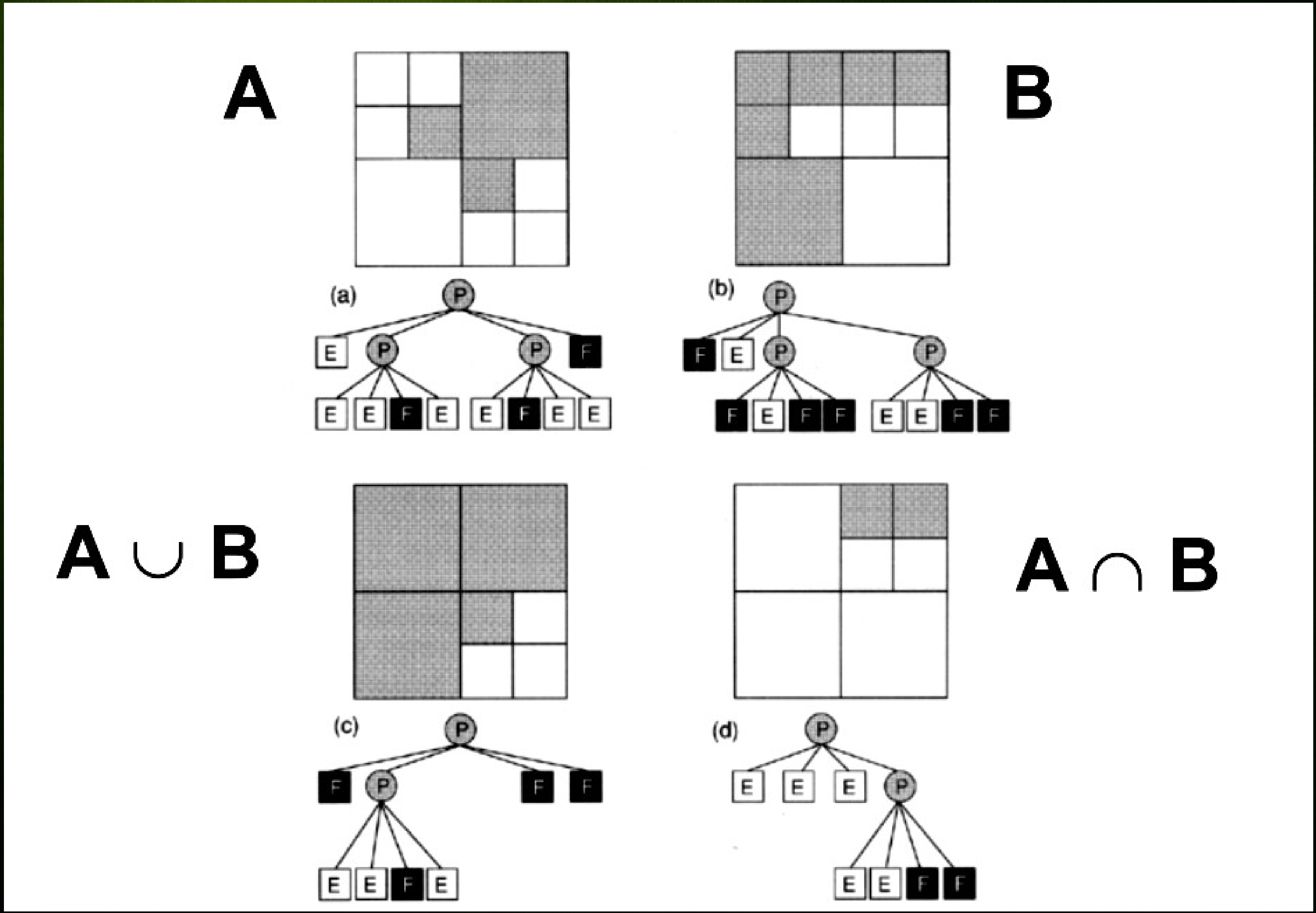


Construction of Octree

- Bottom-Up (merging) scheme
  - Create uniform grid with high resolution
  - For each 8 neighboring cells do
  - If they are all empty (full) merge them into one empty (full) cell, reject sub-cells
  - Otherwise create mixed parent cell and proceed up in the hierarchy



Octree Boolean

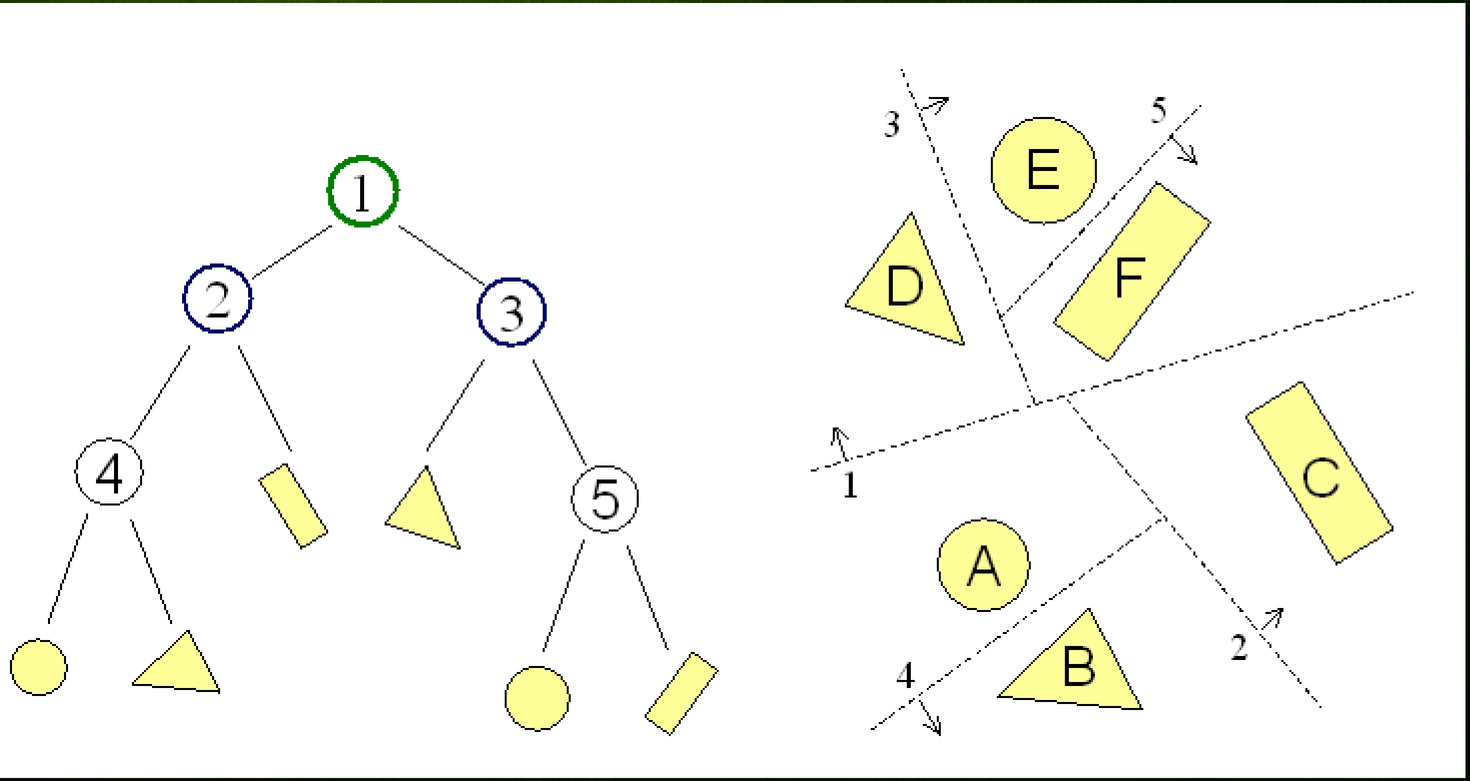


Octree Summary

- Applications
  - Volume data storage (compression)
  - Color quantization
  - Collision detection
- Pros
  - Memory efficient storage
  - Adaptive refinement (more details are preserved)
- Cons
  - Longer point localization (data search)
  - Small change in data → large change in Octree

Binary Space Partition (BSP)

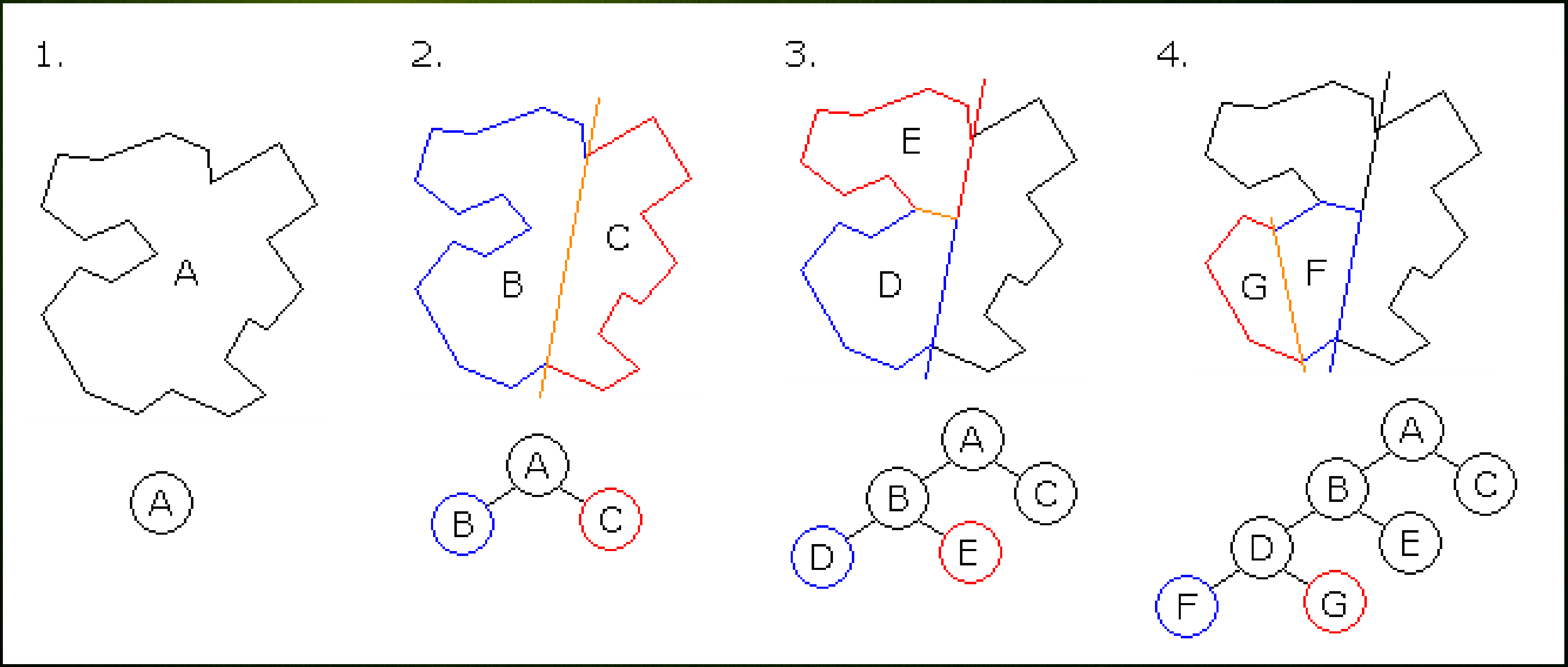
- BSP is a method for recursively subdividing a space into convex sets by hyperplanes
- Every cell is a convex polyhedron



BSP Data Structure

- BSP Node
  - Partitioning hyperplane (position, normal, dist)
  - List of objects (polygons) “intersecting” this node
  - Front child node N_f
  - Back child node N_b

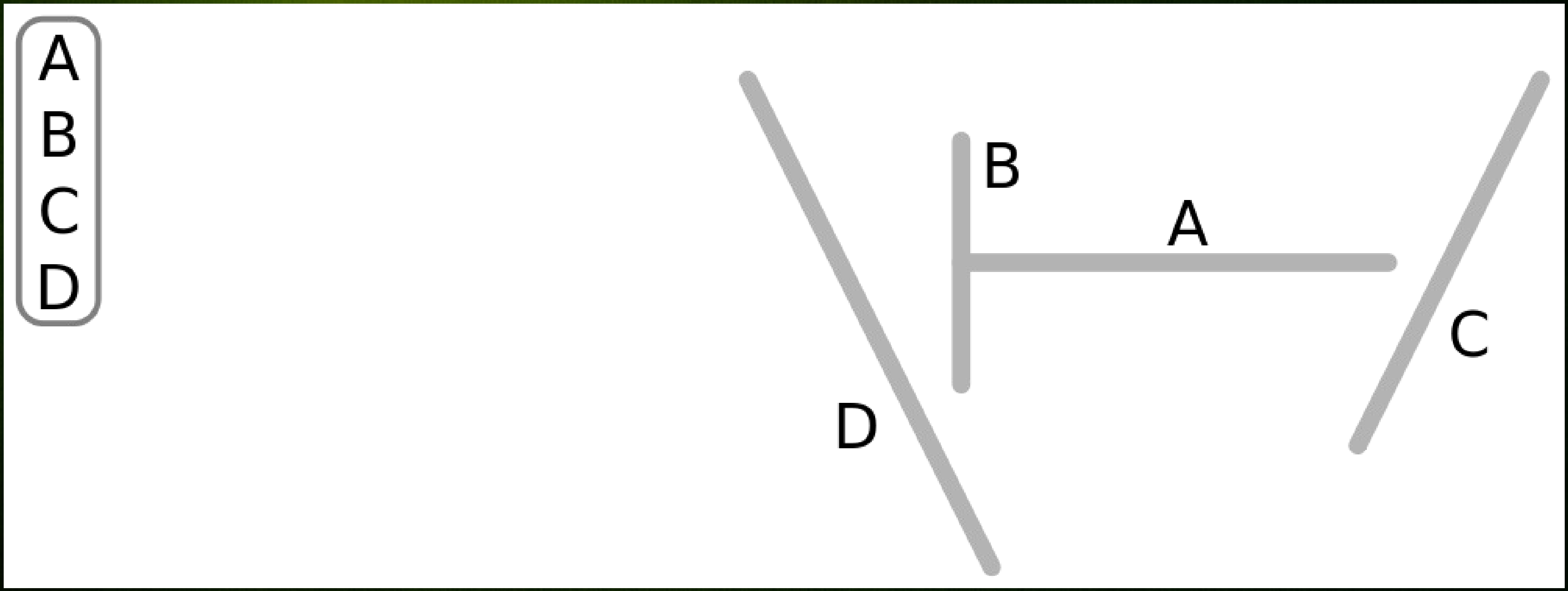
BSP Construction



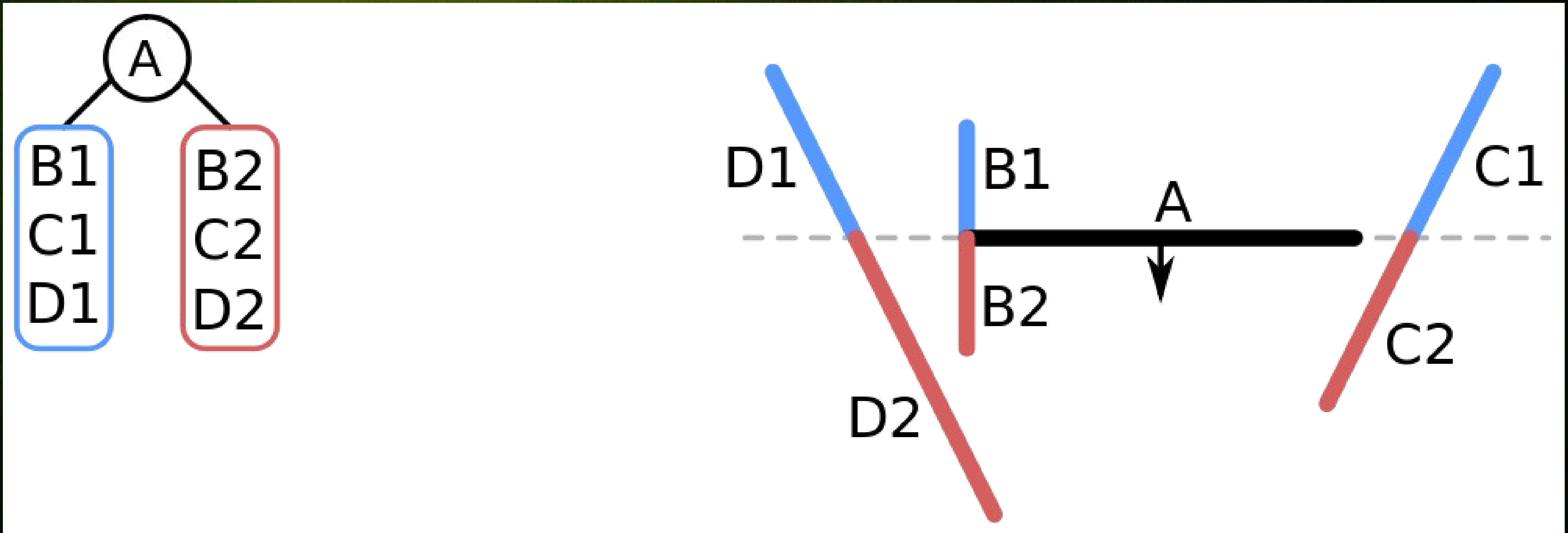
BSP Generation Algorithm

- Choose a polygon P from the list
- Make a node N in the BSP tree, add P to the list of polygons at that node
- For each other polygon Q in the list:
  - If Q is wholly in front (behind) of the plane containing P, move it to the front (back) sub-nodes of P
  - If Q is intersected by the plane containing P, split it into “front” and “back” polygon and move it to respective front and back sub-nodes
  - If Q lies in the plane containing P, add it to the list of polygons at node N
- Repeat this to the list of polygons in front of P
- Repeat this to the list of polygons behind P

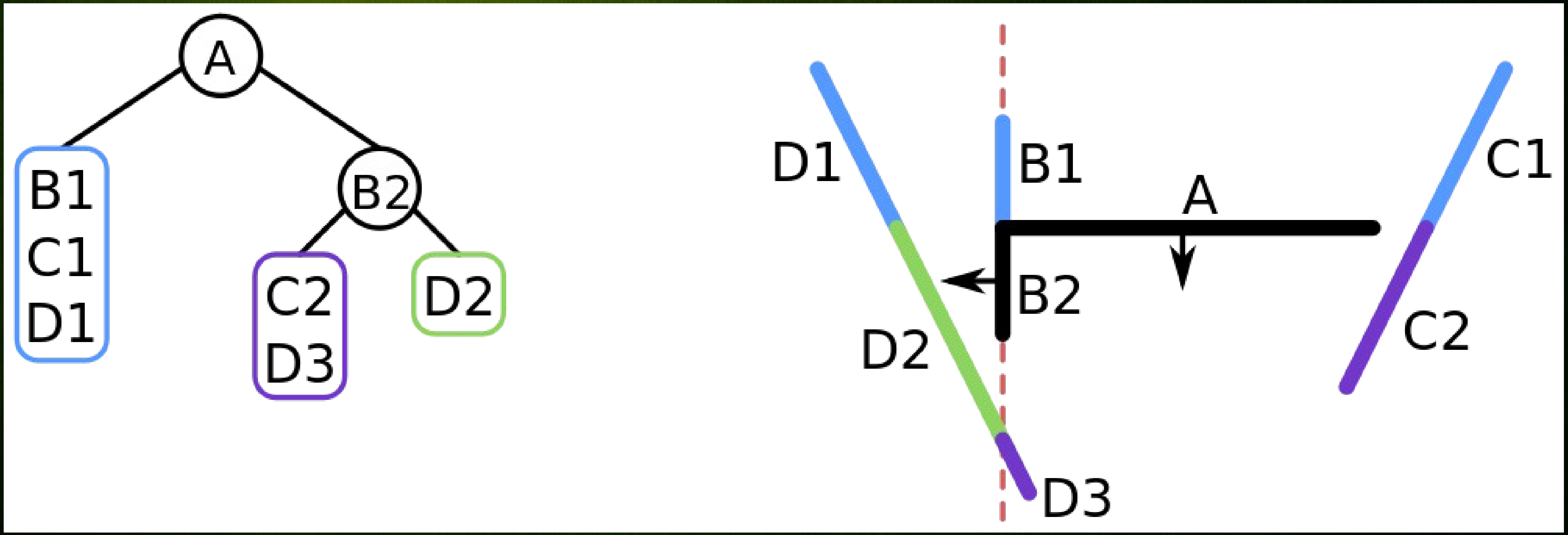
BSP Generation 1



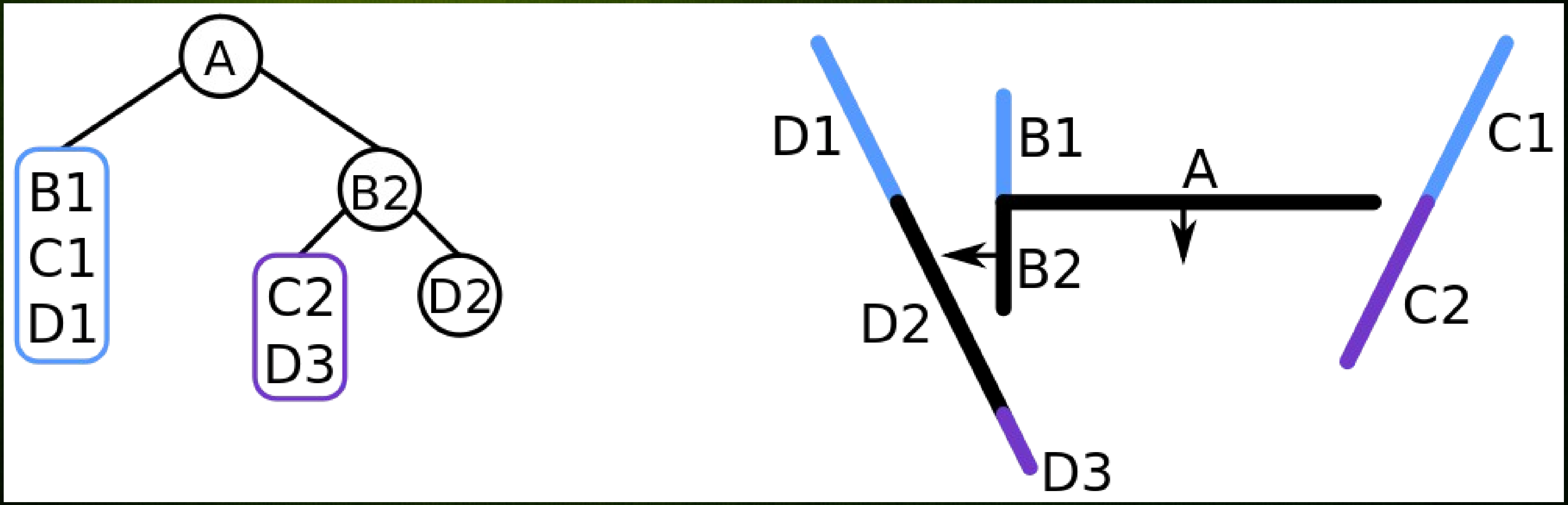
BSP Generation 2



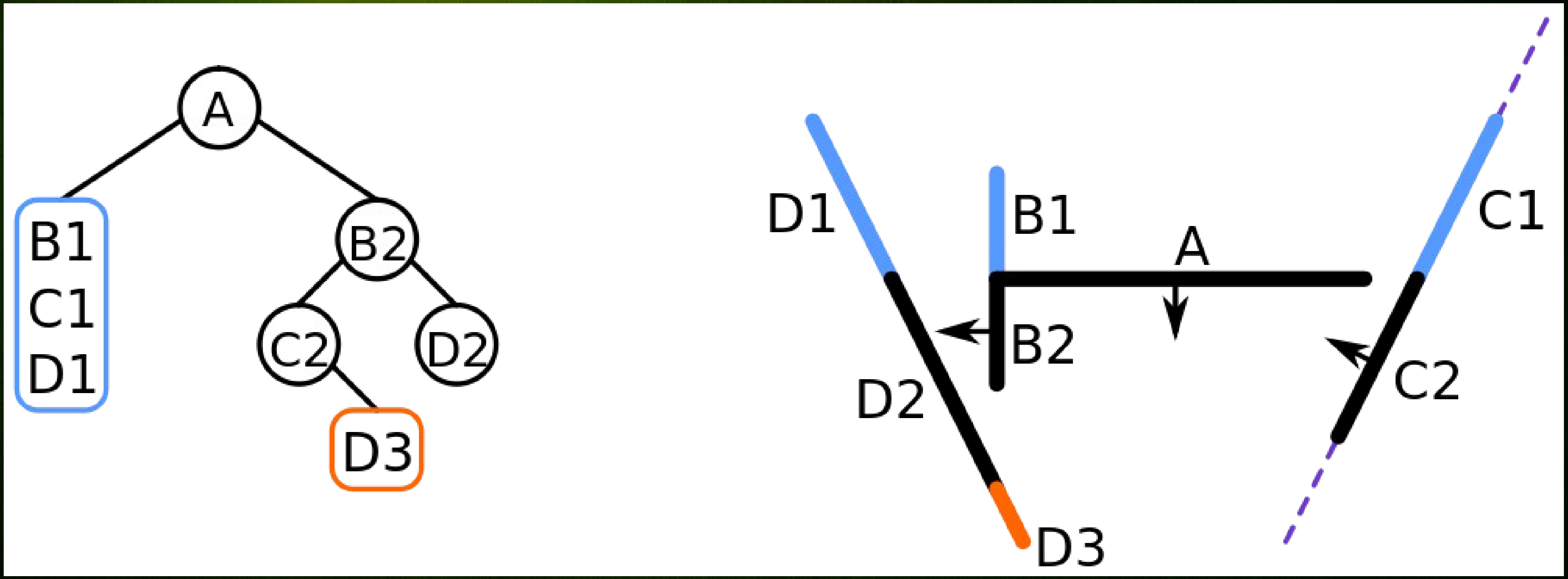
BSP Generation 3



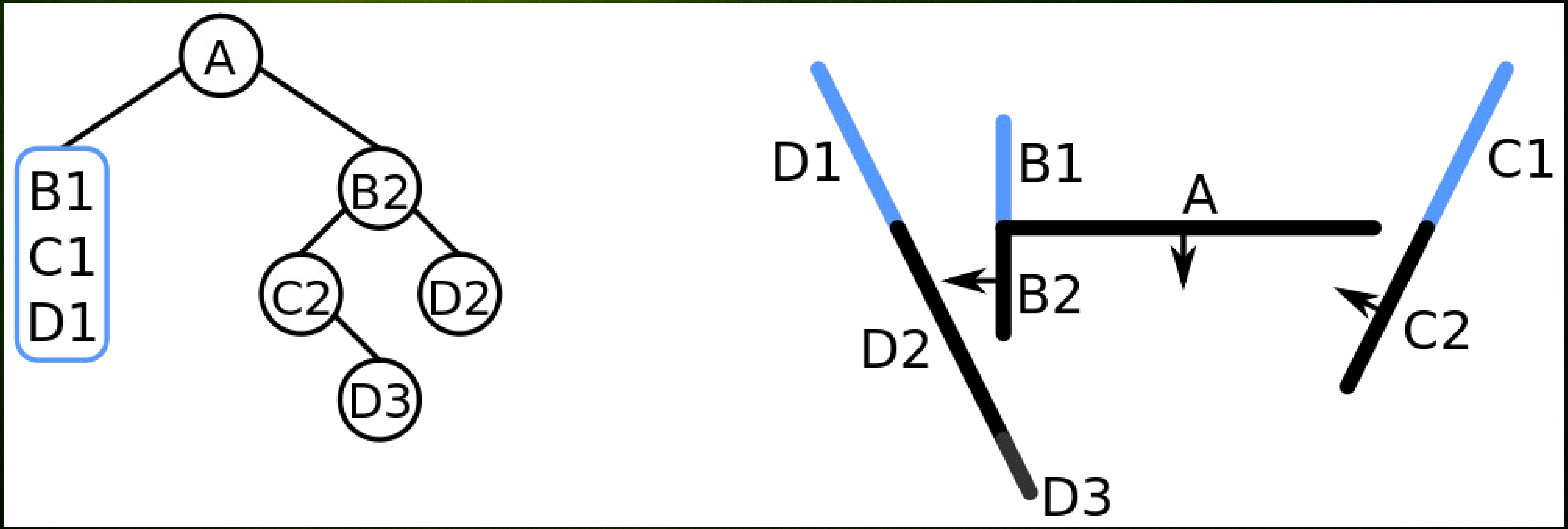
BSP Generation 4



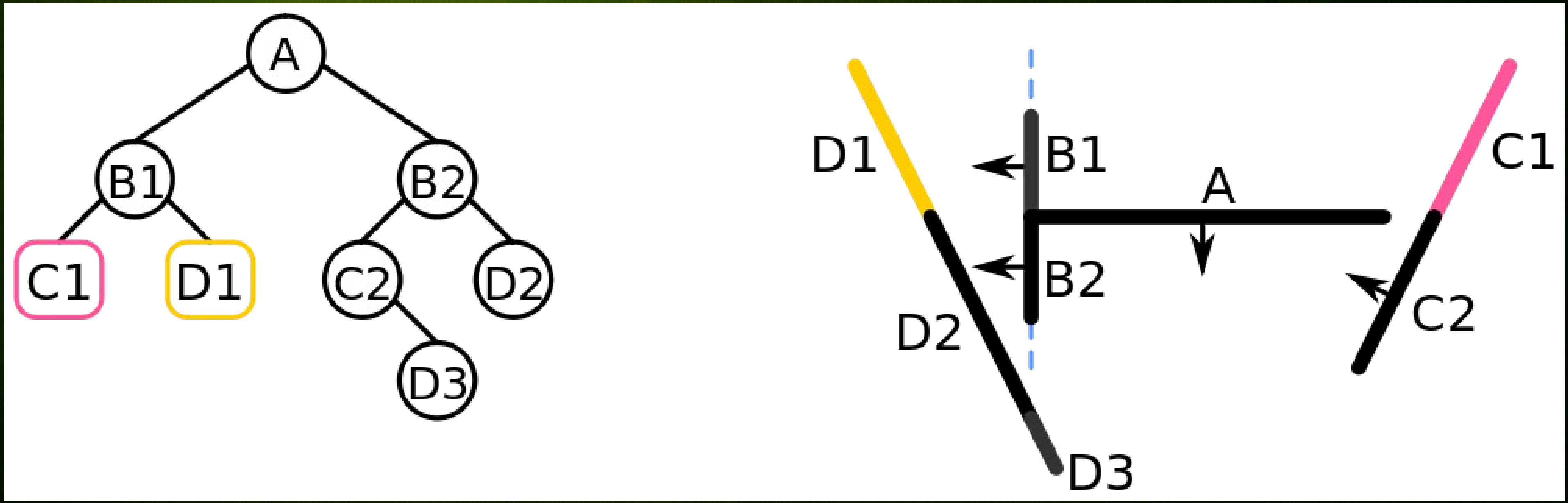
BSP Generation 5



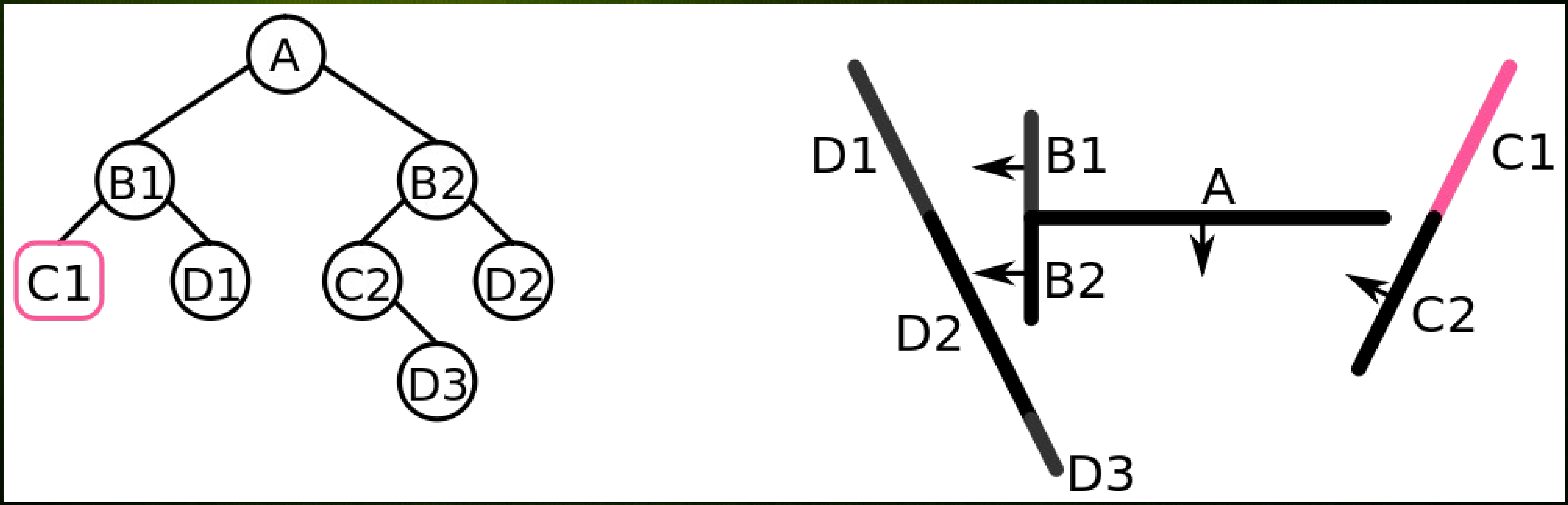
BSP Generation 6



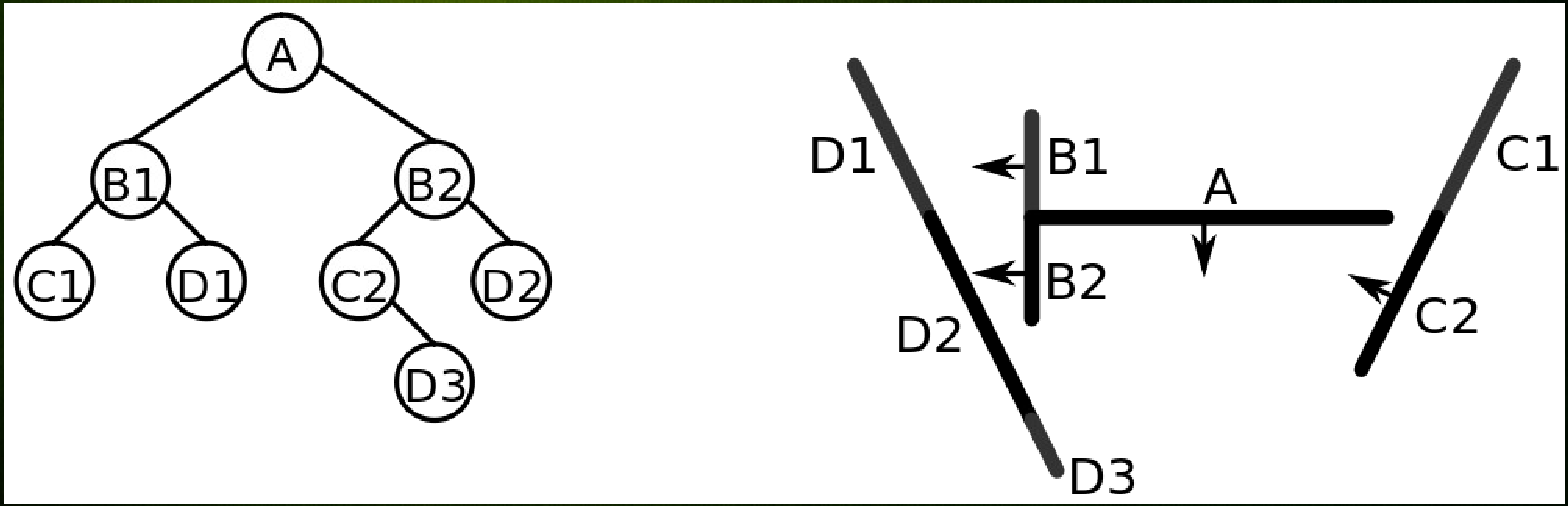
BSP Generation 7



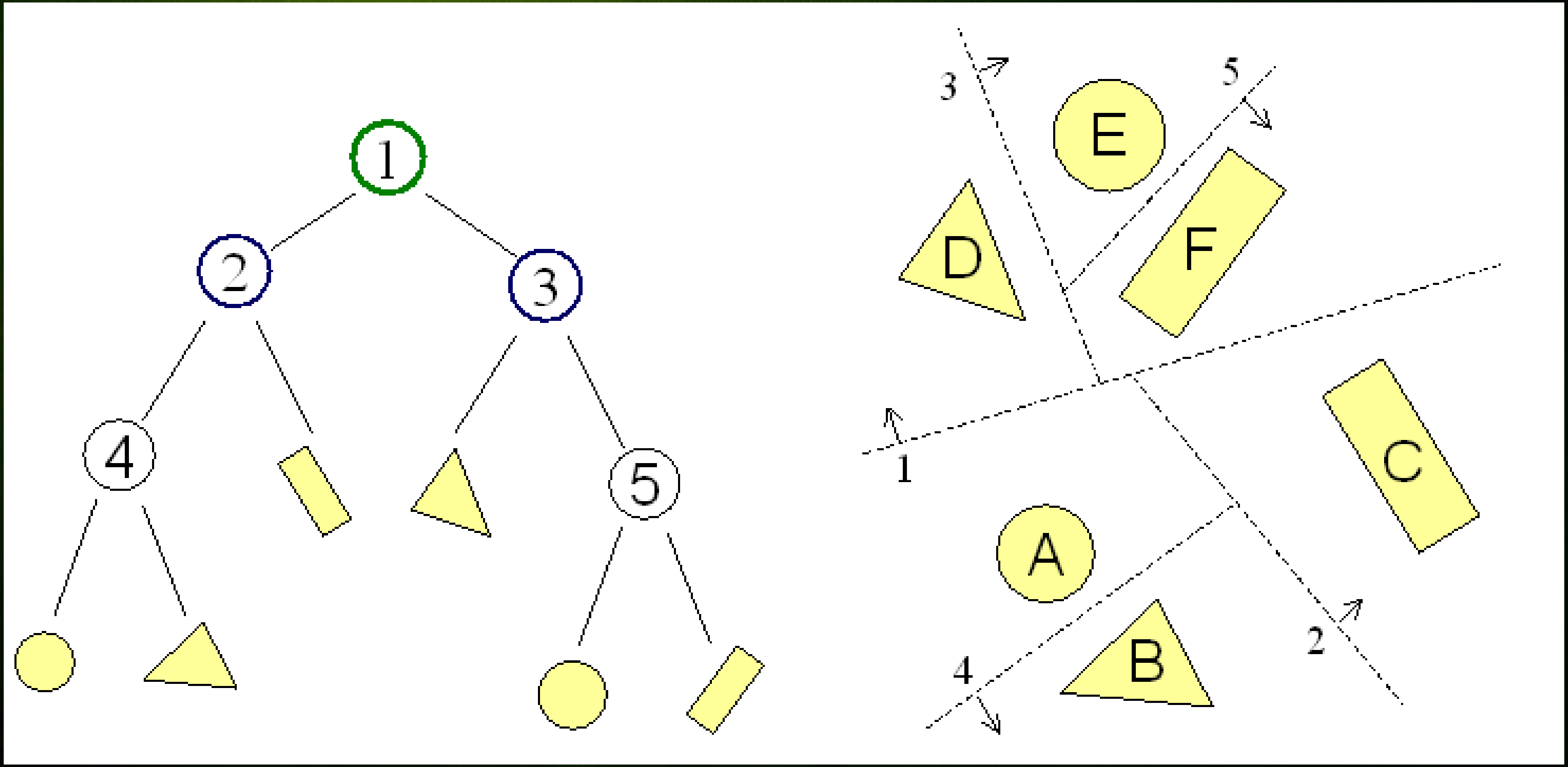
BSP Generation 8



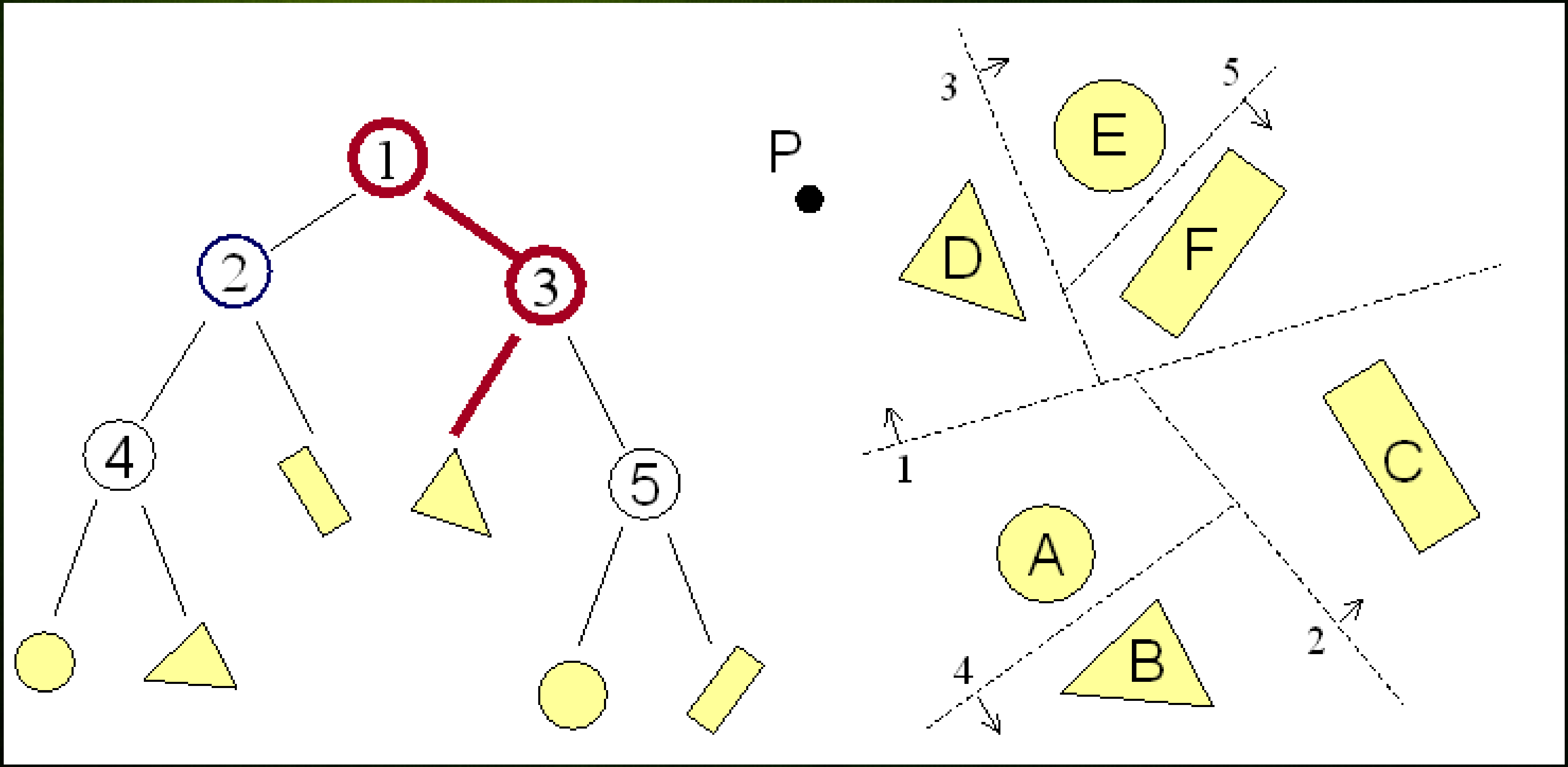
BSP Generation 9



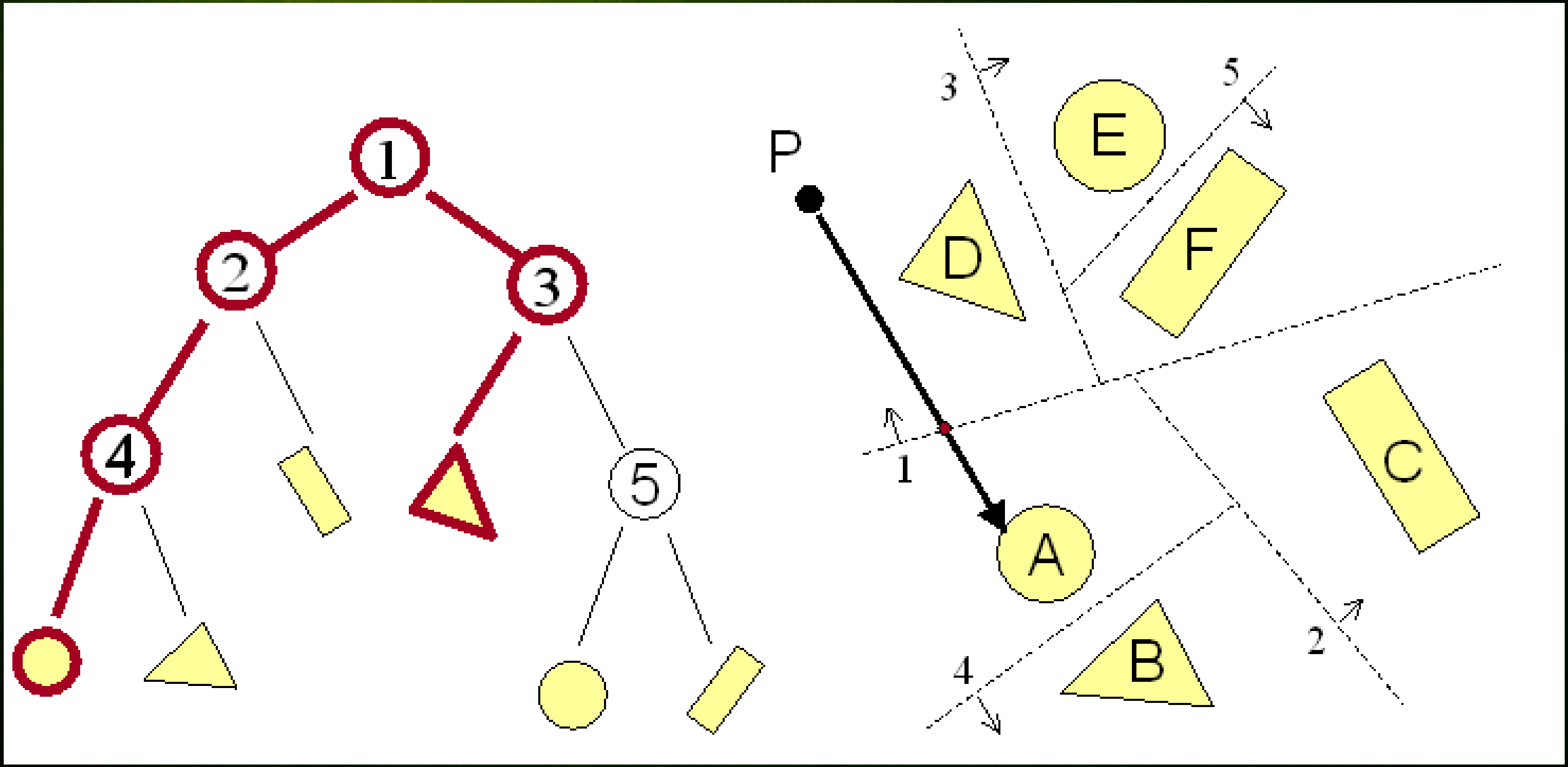
BSP Raytracing 1



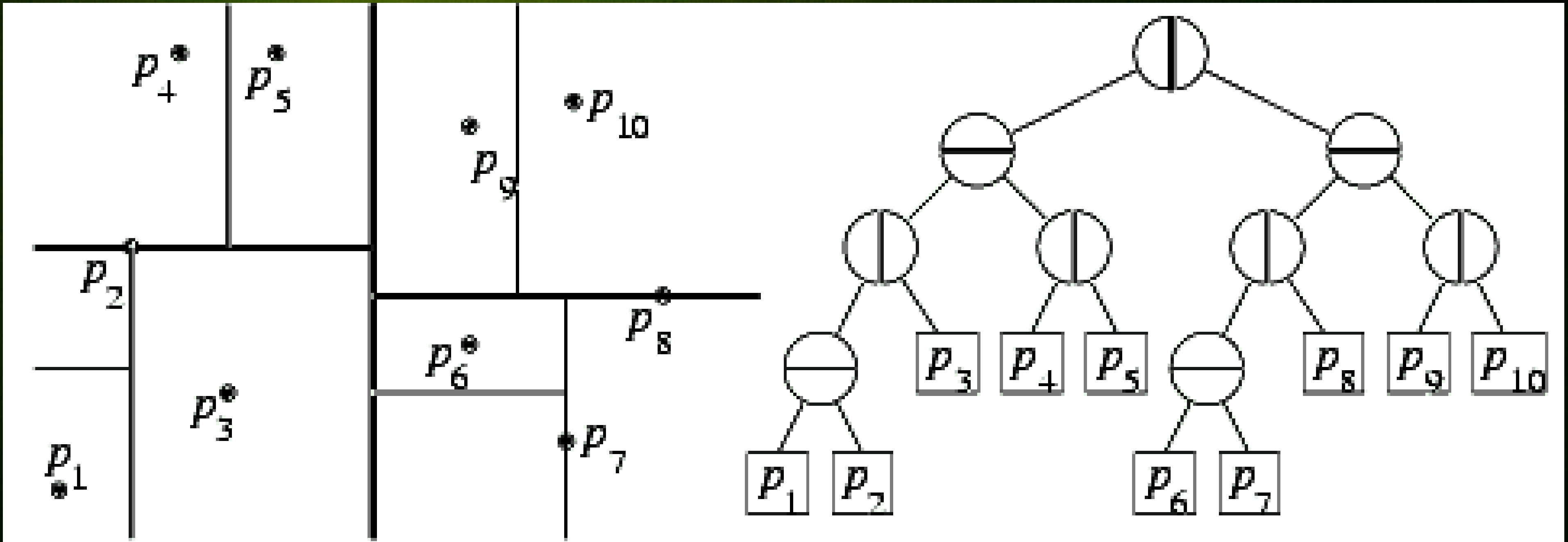
BSP Raytracing 2



BSP Raytracing 3



Orthogonal BSP → kD-tree



Subdivision

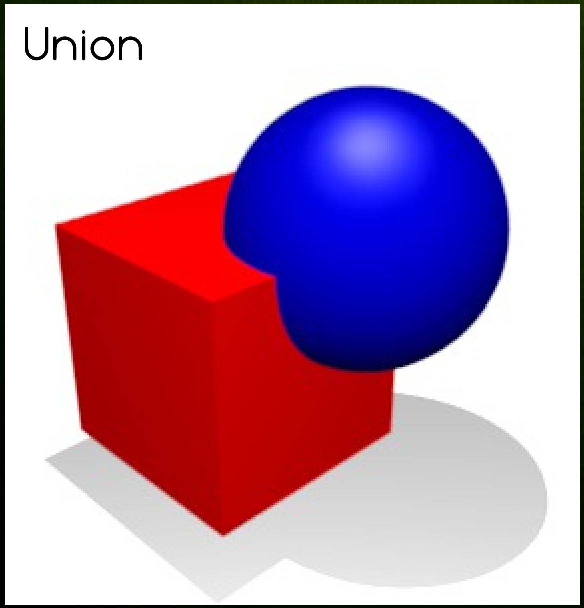
Tree structure

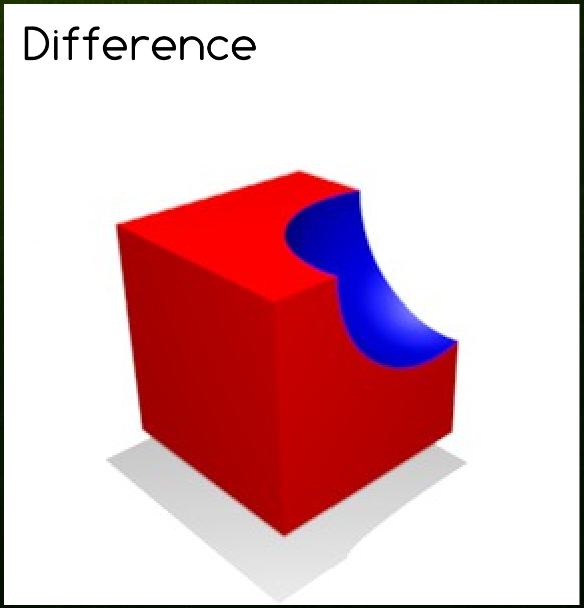
Constructive Solid Geometry

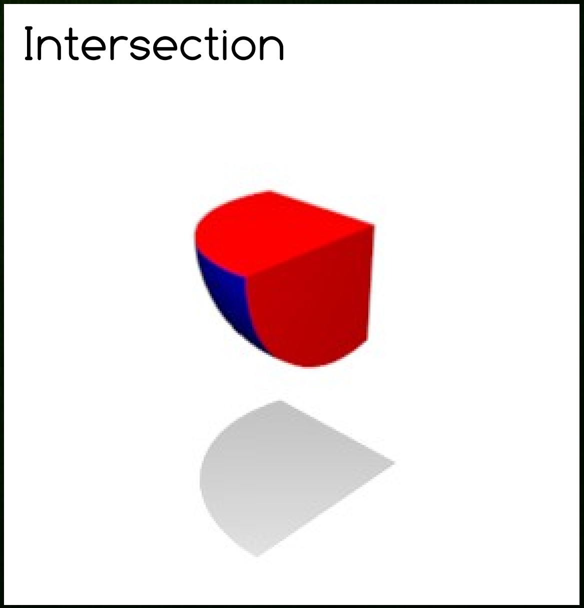
- Constructive Solid Geometry (CSG)
  - Is a volumetric scene representation based on combination of Boolean operations on primitive geometry or other CSG
  - Using only implicitly defined geometry, CSG becomes a a special case of F-Rep
- CSG scene definition includes
  - Primitive geometry objects
  - Tree of Boolean operations

CSG Operations

- Union: $A+B = \{ p \mid p \in A \text{ or } p \in B \}$
- Difference: $A-B = \{ p \mid p \in A \text{ and } p \notin B \}$
- Intersection: $A \wedge B = \{ p \mid p \in A \text{ and } p \in B \}$
- Any other Boolean operation

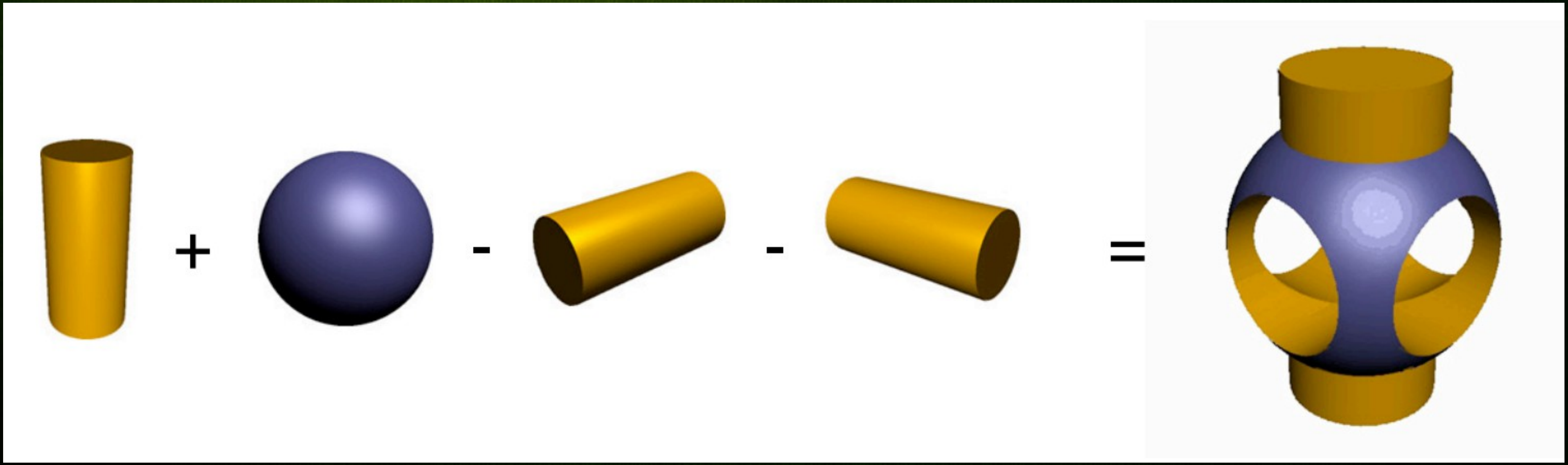




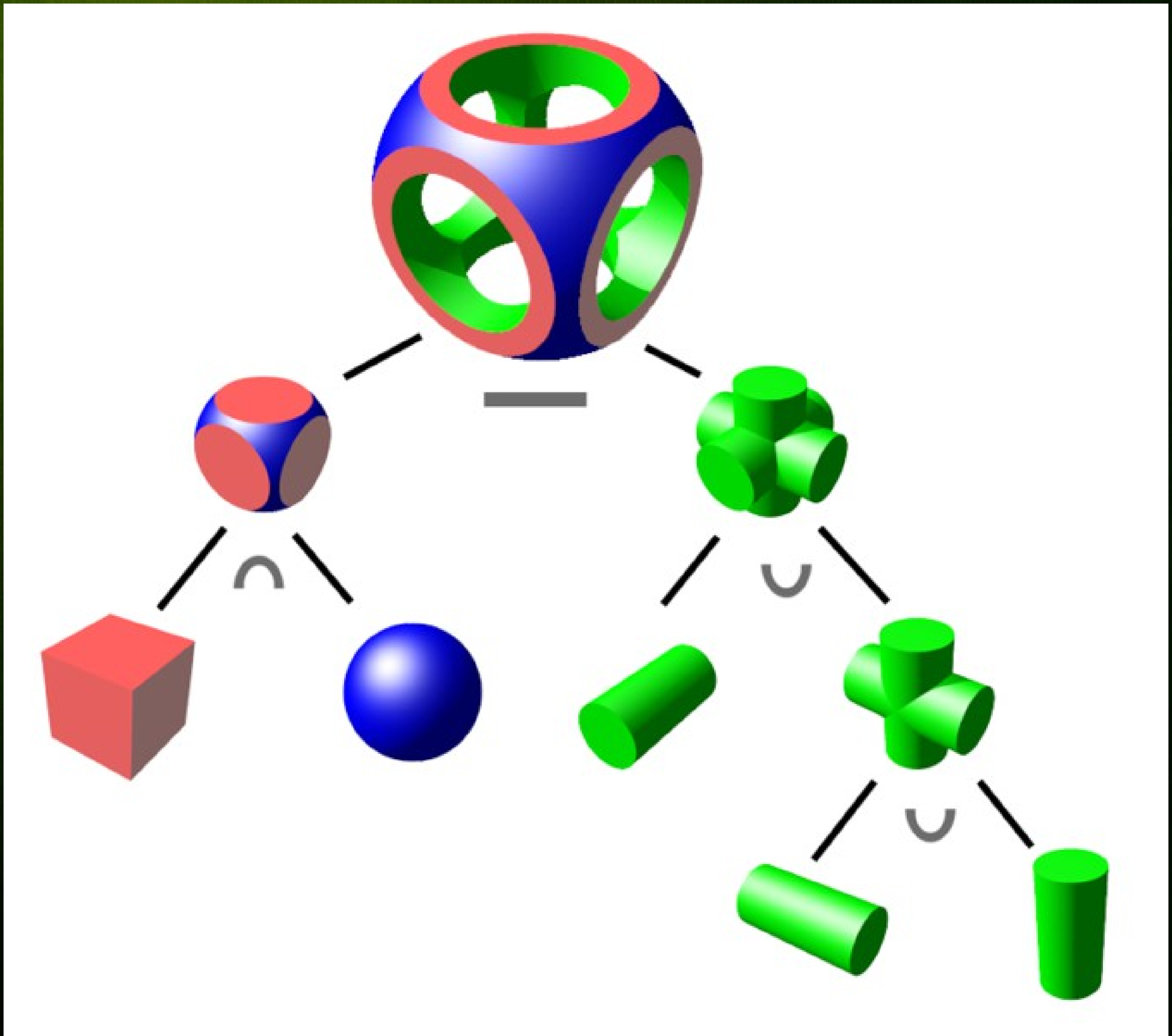


CSG Operations

- Complex objects can be created by applying Boolean operations on primitive geometries in linear order



CSG Operations in hierarchy



CSG Summary

- Applications are mainly in CAD Industry
  - Solid Engineering, Architecture, Security, Army...
- Pros
  - Natural and intuitive modeling strategy
  - Complex shapes can be created from basic shapes
  - Model can always be remodeled
- Cons
  - Using parametric (mesh) primitives can be very slow and complicated
  - Conversion to B-rep can be slow and error-prone

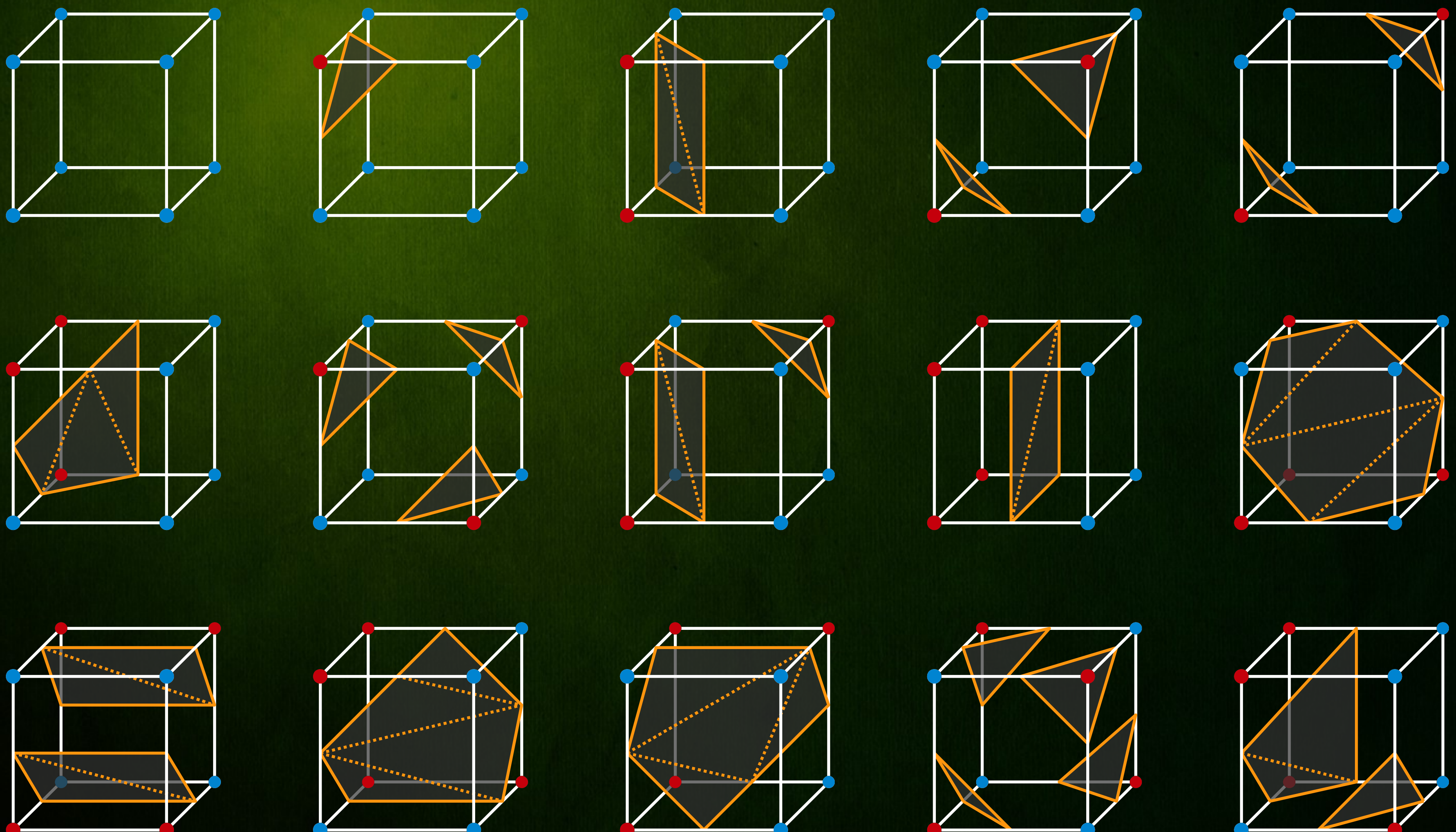
Surface Representation Conversion

- Parametric to Implicit
  - Algebraic solutions
  - Numerical solutions (Scan conversion onto grid)
- Implicit to Parametric
  - Marching Cubes
  - Marching Tetrahedra
  - ...

Marching Cubes Algorithm

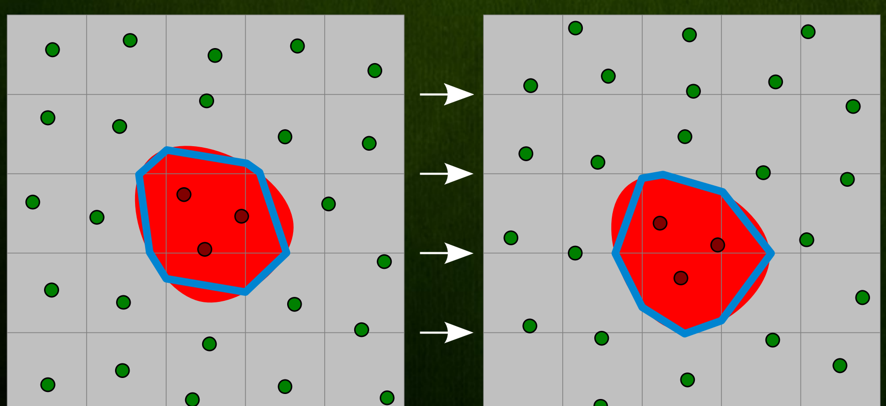
- 1. Specify threshold value
- 2. Decide vertex type (in or out) using the threshold
  - In: value < threshold value
  - Out: value ≥ threshold value
  - If all 8 voxel's vertices are in/out: whole cube is in/out
- 3. Based on 8 vertex states create find MC case in a table and find intersection edges
- 4. Compute vertices coordinates
  - Use linear interpolation with threshold value
- 5. Compute normals
  - Use linear interpolation of vertices normals
  - Normal vector is same as a gradient vector (difference)

Marching Cubes – 15 Cases

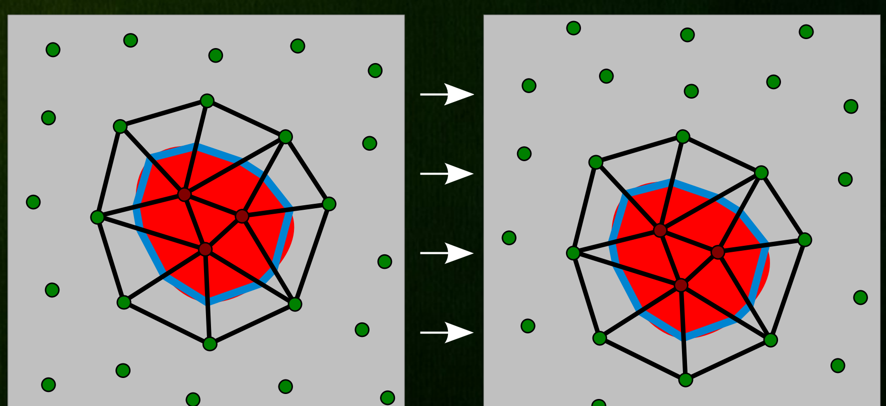


Marching Cubes / Tetrahedra

- Marching cubes produce mesh with stronger turbulence for deforming objects during animation then Marching tetrahedra



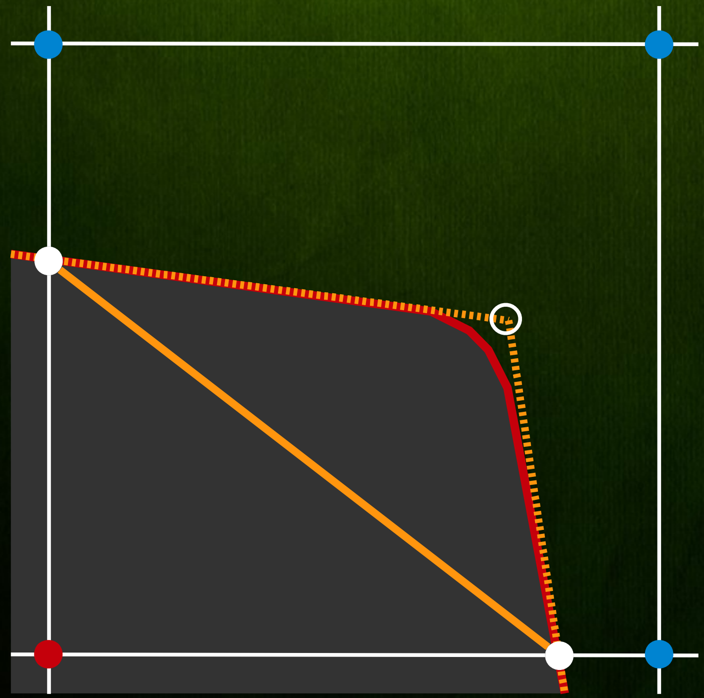
Marching Cubes

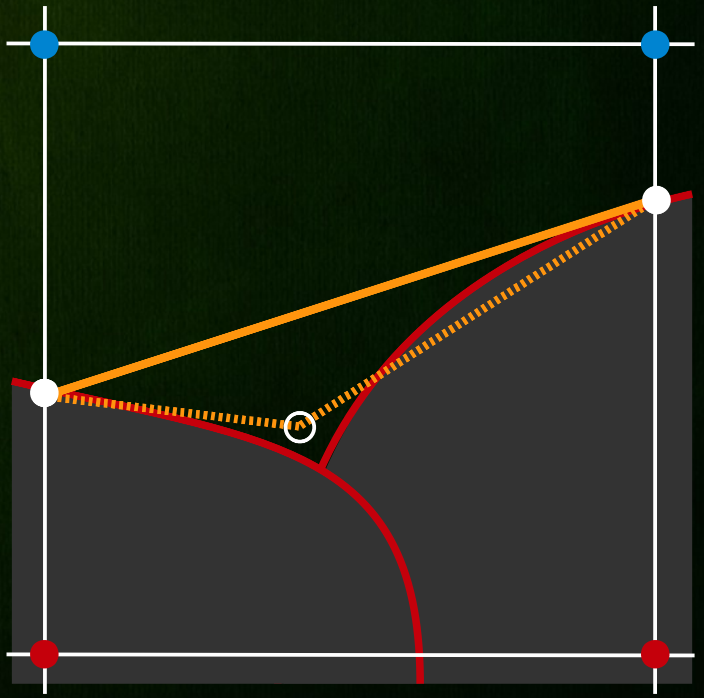


Marching tetrahedra

Marching Cubes - Problems

- Local features are not preserved
- Can be improved when exist using normals and tangent discontinuities





Marching Cubes - Summary

- Applications
  - Trimesh construction for any volume data
  - Remeshing during simulations
  - Surface reconstruction for fluid simulations
- Pros
  - Faster than Marching Tets (no neighbor search)
  - Semi-regular triangulations
- Cons
  - Details are not preserved well
  - Mesh turbulence during animations


the
end
that was enough...