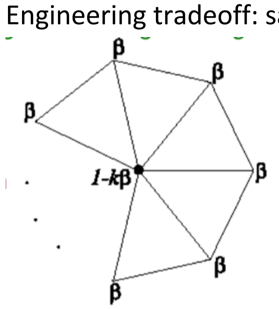
What makes a good surface representation?

- Concise
- Local support – moving one piece only moves that one piece
- Affine invariant – affine transformations do not fundamentally change representation
- Arbitrary topology
- Guaranteed smoothness
- Natural parameterization
- Efficient display
- Efficient intersections

What makes a good parameterization?

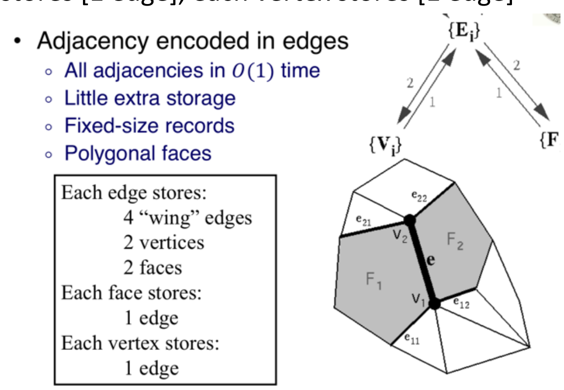
- Above except
- ~~Natural parameterization~~ (doesn't necessarily have to be natural, but not sure?)
- ~~Efficient intersections~~

What is the loop subdivision scheme?

- Refinement step – add one new vertex per edge and connect them to form 4 triangles
- Smoothing step – move new vertex to location based on weighted average of old vertices and neighbors
    - Let k be the number of neighbors and b be the weight. Then position_new = (1-kb)position_original + summation(b*position_i)
    - We can get b by using the original loop equation of warren equation, as long as it guarantees smoothness properties
    - To study the limit, whether it collapses or blows up, we can encode weights into subdivision matrix, exponentiate using eigenvalue decomposition of the subdivision matrix
        - If eigenvalue < 1, then collapses
        - If eigenvalue > 1, then blows up
        - If eigenvalue == -1, then doesn't converge
- Engineering tradeoff: sacrifice interpolation/dof for smoothness
- 

What are 3D representation schemes and their advantages/disadvantages?

- Face table – stores lists of sets of (xyz positions)
    - Pro: naïve solution
    - Con: redundant vertices so moving one vertex is tedious
    - Con: no vertex adjacency info
- Vertex and face table – stores a list of vertices (xyz positions), list of sets of vertices

- Pro: solves redundancy issue
  - Con: no vertex adjacency info
- Adjacency list – store all vertex, edge, face adjacencies
  - Pro: efficient adjacency info
  - Con: extra storage
  - Con: variable sized arrays because we don't know how many neighbors
  - Con: changing the connectivity (e.g. subdividing) is inefficient
- Partial adjacency list – adjacency list with limited info, you derive the rest
  - In the winged edge implementation, each edge stores [4 edges (the wing), 2 vertices, 2 faces], each face stores [1 edge], each vertex stores [1 edge]
  - 

  - Algorithm: choose only edge coming out of v2, iterate counterclockwise around v2 until you cycle back to the start edge
  - Complexity: proportional to size of output

What makes a good spline?

- Local support
- Simple
- Continuous

What is a spline?

- Piecewise polynomial function whose derivative satisfy continuity constraints
- Direct parametric approach to create a curve?
  - Solve for m coefficients of a m-1 parametric function (regression)
  - Con: no local support
  - Con: as the number of points gets larger, the curve oscillates more
  - Con: complex to solve, invert a large system
- Spline approach to create a curve?
  - Rather than having one huge polynomial, piecewise polynomial function... this may not be continuous, but we can just add constraints
  - A spline is a piecewise polynomial function whose derivative satisfy continuity constraints
  - To form a continuity constraint, let the piecewise function go from 0 to 1, representing the start and end. Then C^d_i(1) = C^d_i+1(0). In other words, the derivatives at the endpoint of the previous piece should line up with the derivative of the start for the next one

- Hermite spline and advantages/disadvantages?
  - Use blending functions encoded in matrix to interpolate positions and tangents
  - Pro: C1 continuity
  - Pro: interpolating
  - Con: not C2 continuous
  - Con: requires tangents for control points
  - $$P_k(u) = \underbrace{\begin{pmatrix} u^3 & u^2 & u & 1 \end{pmatrix}}_{\text{parameters}} \underbrace{\begin{pmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}}_{M_{\text{Hermite}}} \underbrace{\begin{pmatrix} p_k \\ p_{k+1} \\ \vec{t}_k \\ \vec{t}_{k+1} \end{pmatrix}}_{\text{boundary info}}$$
  - Parameters * hermite conversion from points/derivatives * points/derivatives
- Cardinal spline and advantages/disadvantages?
  - Gives us tangents for two points based on the surrounding points (4 control points total). Then combine with a Hermite
  - Pro: C1 continuity
  - Pro: interpolating
  - Con: not C2 continuous
  - $$P_k(u) = \begin{pmatrix} u^3 & u^2 & u & 1 \end{pmatrix} \begin{pmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -s & 0 & s & 0 \\ 0 & -s & 0 & s \end{pmatrix} \begin{pmatrix} p_{k-1} \\ p_k \\ p_{k+1} \\ p_{k+2} \end{pmatrix}$$
  - Parameters * hermite conversion from points/derivatives * conversion from old points to old points/new tangents * points
- Uniform Cubic B spline and advantages/disadvantages?
  - Cardinal B spline where you approximate your own points and tangents
  - Pro: C2 continuity
  - Pro: convex hull containment
  - Con: approximating, not interpolating
  - Con: missing end points
  - $$P_k(u) = \begin{pmatrix} u^3 & u^2 & u & 1 \end{pmatrix} \begin{pmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \frac{1}{6} \begin{pmatrix} 1 & 4 & 1 & 0 \\ 0 & 1 & 4 & 1 \\ -6s & 0 & 6s & 0 \\ 1 & -6s & 0 & 6s \end{pmatrix} \begin{pmatrix} p_{k-1} \\ p_k \\ p_{k+1} \\ p_{k+2} \end{pmatrix}$$
  - Parameters * hermite conversion from points/derivatives * conversion from old points to new points/new tangents * points

Properties:

- Translation Commutativity:
  $BF_0(u) + BF_1(u) + BF_2(u) + BF_3(u) = 1$ for all $0 \le u \le 1$.
- Continuity:
  $0 = BF_0(1),\ BF_0(0) = BF_1(1),\ BF_1(0) = BF_2(1),\ BF_2(0) = BF_3(1),\ BF_3(0) = 0$
- Convex Hull Containment:
  $BF_0(u), BF_1(u), BF_2(u), BF_3(u) \ge 0$, for all $0 \le u \le 1$.
- Interpolation:
  - We want the spline segments to satisfy:

$$P_k(0) = p_k \quad \text{and} \quad P_k(1) = p_{k+1}$$

⇒ At the end-points, the blending functions satisfy:

$$\begin{matrix} BF_0(0) \\ BF_1(0) \\ BF_2(0) \\ BF_3(0) \end{matrix} = \begin{matrix} 0 \\ 1 \\ 0 \\ 0 \end{matrix} \quad \text{and} \quad \begin{matrix} BF_0(1) \\ BF_1(1) \\ BF_2(1) \\ BF_3(1) \end{matrix} = \begin{matrix} 0 \\ 0 \\ 1 \\ 0 \end{matrix}$$

3D surface splines properties?

- Translation commutativity
- Continuity
- Convex hull containment
- Interpolation

Tensor product spline?

- Using matrices to get surface spline

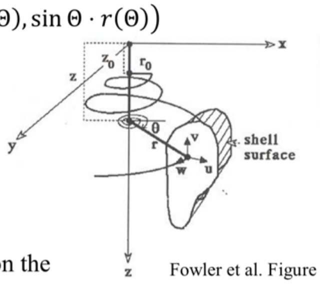
Types of procedural modeling?

- Sweeps – sweep a curve about a line (which defines rotation, translation, scale)

Helico-Spiral definition:

$$H(\Theta) = \left(\cos\Theta \cdot r(\Theta), z(\Theta), \sin\Theta \cdot r(\Theta)\right)$$

Angle: Θ
Radius: $r(\Theta) = e^{\lambda\Theta}$
Height: $z(\Theta) = e^{\mu\Theta}$



Fowler et al. Figure 1

Instead, we compute a local frame $\left(u(\Theta), v(\Theta), w(\Theta)\right)$ at each point on the sweep curve and describe the surface w.r.t. this frame:

$$S(\Theta, \Psi) = H(\Theta) + \left(u(\Theta) \cdot C_u(\Psi) + v(\Theta) \cdot C_v(\Psi)\right) \cdot r(\Theta)$$

- Fractals – can either be a normal fractor, or statistically similar (initiator shape, replace subshapes with self-similar random pattern), but the idea is that you replace subshapes
- Grammars – using a grammar, replace words until there is nothing to replace

Solid modeling and advantages/disadvantages?

- Pro: easy to visualize in graphics hardware

- Con: some models cannot be represented with boundary
- Con: Difficult to intersect models

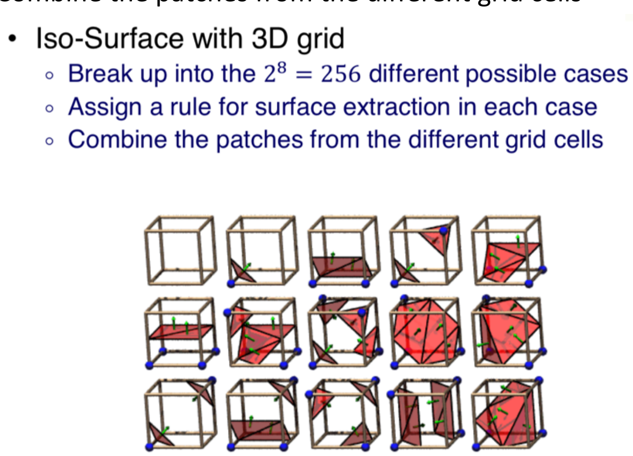
Implicit surfaces?

- Using a 3 dimensional function, good for simple shapes, e.g. quadrics, but has been shown that we can sum gaussians to create a good blobby model
- Pro: easy to test if a point is on or inside the surface
- Pro: easy to intersect surfaces
- Con: hard to describe complex shapes and functions
- Con: hard to enumerate points on surface
- Blobby model

Voxels?

- Grid representation, think Minecraft
- Can be binary model (either there is a block or not) or continuous (encode information, e.g. color)
- Voxel display – slicing, raycast for density, iso-surface extraction using marching cubes algorithm
- Pro: simple
- Pro: same complexity for all objects
- Pro: natural acquisition for some applications
- Pro: boolean operations are trivial
- Con: approximate
- Con: not affine invariant
- Con: large storage requirement
- Con: expensive to display
- Con: fixed resolution

Marching cubes algorithm?

- Creates a smoother interpretation of voxel grid
- Define 2^8 = 256 different possible cases
- Assign a rule for surface extraction in each case
- Combine the patches from the different grid cells
- 

- Cons: May contain ambiguities

Quadtree/Octtree?

- Same idea as voxels, but refine resolution based on what you need
- How to handle zero-crossings? Use info from side with more information

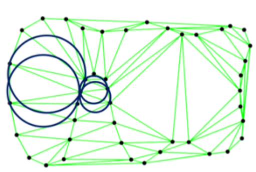
Range processing?

- Manual initial alignment
- ICP to an existing scan
    - For each point in scan1, find the nearest point in scan2
    - Translate/rotate to minimize distance between two points
- Global relaxation
- Merging using volumetric method

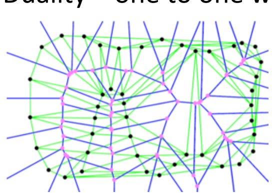
Define convex, convex hull?

- Convex: line segment connecting any two points is also in the polygon
- Convex hull: smallest convex set containing set

Delaunay triangulation?

- Circumscribing circles do not contain any other points
- Compactness property – maximizes minimum angle
- 

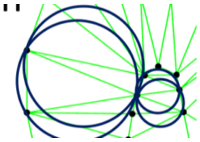
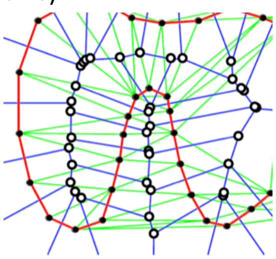
Voronoi diagram of set S?

- Partitions space into regions such that all points are closer to partition than any other point in S
- Vertices are equidistant from 3+ points in incident cells
- For a vertex, we can draw an empty circle that touches the point in 3+ incident cells
- Duality – one to one with Delaunay triangulation
- 

Medial axis

- Set of circles/spheres that only meet the shape tangentially at 2+ points.
- For a reasonable point sample, well sampled by Voronoi diagram

Space partitioning – you in or out?

- Spectral partitioning (organized points)
    - Assign weight to each edge indicating if two triangles are likely to have the same label (can set weights based on how much two circles overlap)
    - Find a partition into roughly even pieces minimizing sum of weights along the partition
    - 
- Crust algorithm (organized points)
    - Compute Delaunay triangulation
    - Compute Voronoi vertices
    - Keep edges for which there is a circle that contains edge but not voronoi vertices (do not cross medial axis)
    - 
- Implicit surface reconstruction from unorganized points
    - Local signed distance function
    - Extract zero level set
    - Get normals by fitting a line to the neighbors at each point, and build a Euclidian MST and propagate orientation from root
- Poisson reconstruction
    - Transform sample into (least-squares) vector field
    - Fit scalar-field to gradients
    - Extract isosurface

In-betweening?

- Interpolate/approximate transformations rather than positions
- Can use spline
- Curve normalization
    - Con: ambiguity (line that passes through origin)
    - Con: uniform sample on original curve not necessarily uniform on new curve
    - SLERP – parameterize using angle, calculate point, blend parameters and evaluate
- SVD factorization
- Euler angles, then interpolate
- Quaternions [MAKE SURE TO STUDY THIS]
- Exponential/Logarithmic map with skew symmetric matrices

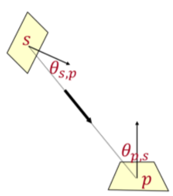
Kinematics versus dynamics?

- Kinematics considers only motion

- Forward – given two angles of an arm, find end effector (hand)
- Inverse – given end effector (hand), find two angles of arm. May have multiple or no solutions, so we may have to minimize energy or perform non-linear optimization aka more computation

- Dynamics considers underlying forces, physics simulation
    - Discretize time steps, minimize, solve linear or nonlinear iterative optimization technique
    - Pro: Animator doesn't have to think about physics
    - Pro: easy to vary motions due to new parameters and/or constraints
    - Con: must specify constraints/objective functions
    - Con: need to avoid local minima during optimization

Radiosity?

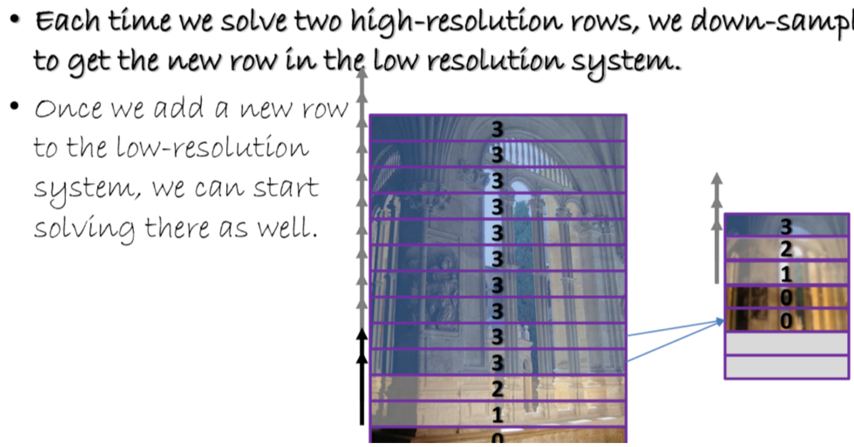
- Assumes that lights are uniformly emissive, and that objects are Lambertian
- We need to account for distance and angles between surfaces
- 

$$\boxed{B(p) = E(p) + \rho(p) \int_S V(s,p) \cdot \frac{\cos\theta_{s,p} \cdot \cos\theta_{p,s}}{\|s-p\|^2} \cdot B(s)ds}$$

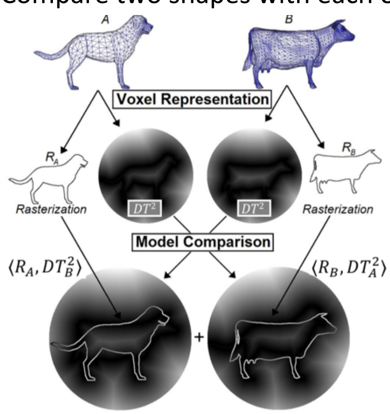
- Emissivity + material property * integral of visibility and radiosity assumptions and brightness

Gradient domain?

- Rather than representing an image, represent image as a disjoint set of pixel value differences
- E.g. in a blank image, no gradient, but in stripe.bmp, gradients going between black/white
- Good for stitching images together, HDR compression
- We would like to get pixel values from horizontal and vertical gradients, but over constrained problem, may have multiple solutions.
- However, rather than setting differences to 0, we can minimize instead.
- Gauss Seidel Iteration algorithm?
    - Runs over each pixels, calculates D^T D Lapalacian * pixels = gradients
- Multi-grid solver algorithm?
    - A few solver iterations on high-res
    - Compute residual and downsample
    - Solve low-res (recursive)
    - Upsample and add to high-res
    - A few solver iterations on high-res
    - Parallelization optimization tricks:



Shape matching general approach using Euclidian distance transform?

- Euclidian distance transform basically places right cones w/ apices at surface boundary, representing distance falling off linearly. Rasterizes to get a map of depth
- Compare two shapes with each others' distance transforms with dot product
- 

- Pro: discriminating
- Pro: quick to compute
- Pro: matching over rigid body transformations
- Con: partial object matching
- Con: difficult for articulated figures and deformable models