Issues

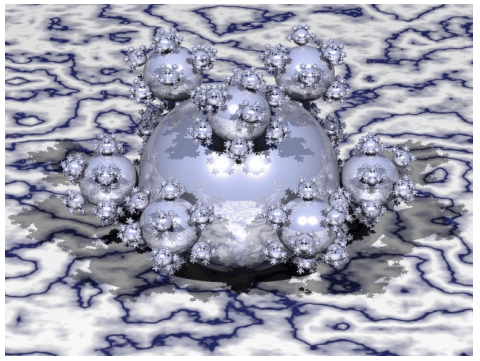
- Sampling (aliasing)
- Very large numbers of objects
  - Need making intersections efficient, exclude as much as possible using clever data structures
- Surface detail
  - bumps, texture, etc.
- Illumination effects
  - Caustics, specular to diffuse transfer
- Camera models

Sampling

- Simplest ray-tracer is one ray per pixel
  - This gives aliasing problems
- Solutions
  - Cast multiple rays per pixel, and use a weighted average
  - Rays can be on a uniform grid
  - It turns out to be better if they are “quite random” in position
    - “hard-core” Poisson model appears to be very good
    - different patterns of rays at each pixel

Efficiency - large numbers of objects

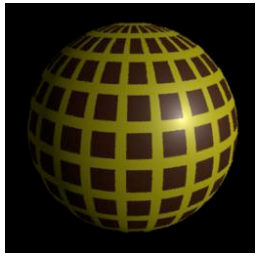
- Construct a space subdivision hierarchy of some form to make it easier to tell which objects a ray might intersect
- Uniform grid
  - easy, but many cells
- Bounding Spheres
  - easy intersections first
- Octtree
  - rather like a grid, but hierarchical
- BSP tree



500,000 spheres, Henrik Jensen, http://www.gk.dtu.dk/~hwj

Surface detail

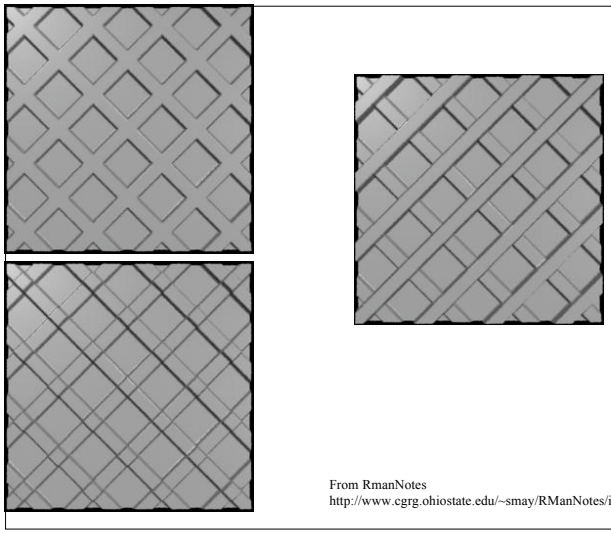
- Knowing the intersection point gives us a position in intrinsic coordinates on the surface
  - e.g. for a triangle, distance from two of three bounding planes
- This is powerful - we could attach some effect at that point
- Texture maps:
  - Make albedo (or color) a function of position in these coordinates
  - Rendering: when intersection is found, compute coordinates and get albedo from a map
  - This is not specific to ray-tracing



From RmanNotes: http://www.cgrg.ohio-state.edu/~smay/RManNotes/index.html

Surface detail, II

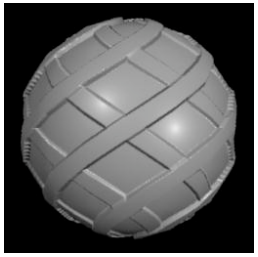
- Bumps
  - we assume that the surface has a set of bumps on it
    - e.g. the pores on an orange
  - these bumps are at a fine scale, so don't really affect the point of intersection, but do affect the normal
  - strategy:
    - obtain normal from "bump function"
    - shade using this modified normal
    - notice that some points on the surface may be entirely dark
    - bump maps might come from pictures (like texture maps)



From RmanNotes
http://www.cgrg.ohiostate.edu/~smay/RManNotes/index.html

Surface detail, III

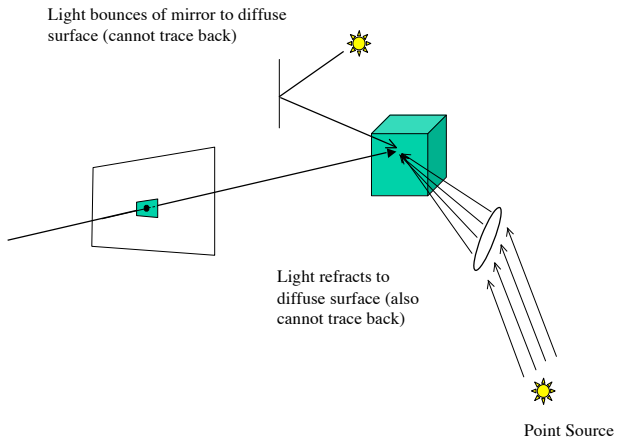
- A more expensive trick is to have a map which includes **displacements** as well
- Must be done **before** visibility



From RmanNotes: http://www.cgrg.ohio-state.edu/~smay/RManNotes/index.html

Illumination effects

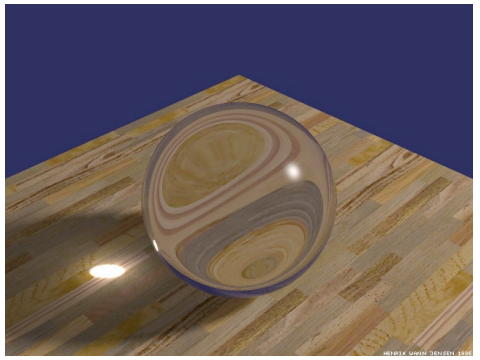
- Caustics:
  - refraction or reflection causes light to be "collected" in some regions.
- Specular-> diffuse transfer
  - source reflected in a mirror
- Can't render this by tracing rays from the eye - how do they know how to get back to the source?



Illumination effects (cont)

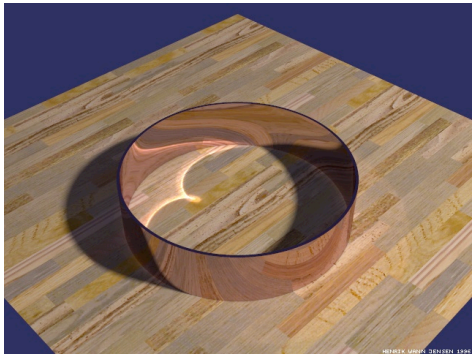
- To get the effect of light reflected and refracted from sources onto diffuse surfaces, we can trace rays **from** the light **to** the first diffuse surface
  - leave a note that illumination has arrived - an illumination map, or photon map
  - sometimes referred to as the forward ray
  - now retrieve this note by tracing eye rays
- Issues
  - efficiency (why trace rays to things that might be invisible?)
  - aliasing (rays are spread out by, say, curved mirrors)

Refraction caustic



Henrik Jensen, http://www.gk.dtu.dk/~hwj

Reflection caustic



Henrik Jensen, http://www.gk.dtu.dk/~hwj

Refraction caustics

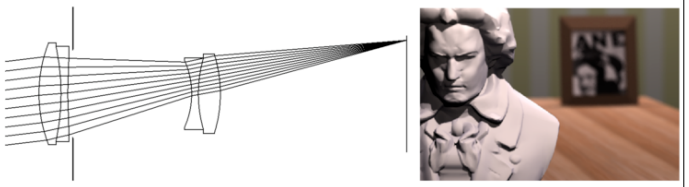
Henrik Jensen, http://www.gk.dtu.dk/~hwj

Lens Effects

Note that a ray tracer very elegantly deals with the projection geometry that we struggled with in earlier lectures which was based on a very simple and "ideal" camera model

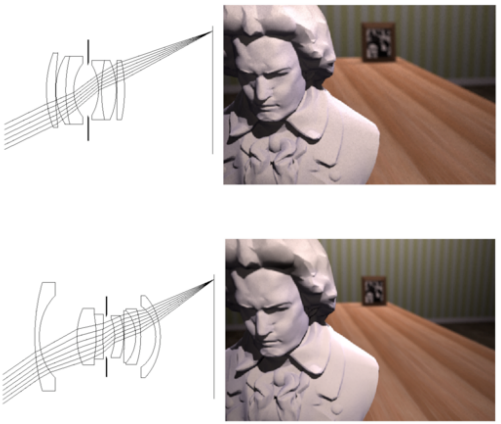
We can go further and introduce a more interesting or realistic camera model

from
A Realistic Camera Model for Computer Graphics
Craig Kolb, Don Mitchell, and Pat Hanrahan
Computer Graphics (Proceedings of SIGGRAPH '95), ACM SIGGRAPH, 1995, pp. 317-324



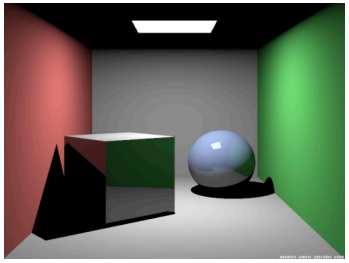
Note limited depth of field, just like a real lens

from
A Realistic Camera Model for Computer Graphics
Craig Kolb, Don Mitchell, and Pat Hanrahan
Computer Graphics (Proceedings of SIGGRAPH '95), ACM SIGGRAPH, 1995, pp. 317-324

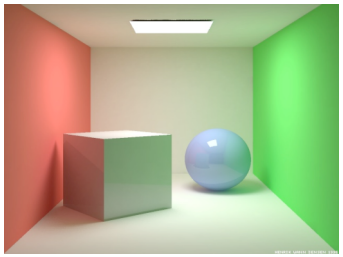


from
A Realistic Camera Model for Computer Graphics
Craig Kolb, Don Mitchell, and Pat Hanrahan
Computer Graphics (Proceedings of SIGGRAPH '95), ACM SIGGRAPH, 1995, pp. 317-324

Radiosity



Ray-traced Cornell box, due to Henrik Jensen, http://www.gk.dtu.dk/~hwj



Radiosity Cornell box, due to Henrik Jensen, http://www.gk.dtu.dk/~hwj, rendered with ray tracer

Radiosity

Want to capture the basic effect that surfaces illuminate each other

Again, following every piece of light from a diffuse reflector is impractical--but combinations of brute force and clever hacks can be done

Another approach: Radiosity methods

Radiosity

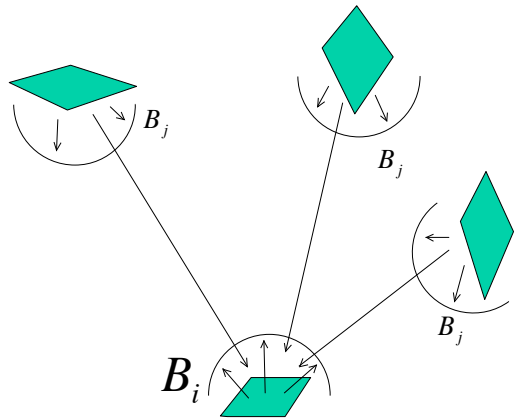
Think of the "world" as a bunch of patches. Some are sources, (and reflect), some just reflect. Each sends light towards all the others.

Consider one color band at a time (some of the computation is shared among bands).

Each surface, i, *radiates* reflected light, B_i

Each surface, *emits* light E_i (if it is not a source, this is 0).

Denote the albedo of surface i as ρ_i



Radiosity equation

$$B_i = E_i + \rho_i \sum_j F_{j \to i} B_j \frac{A_j}{A_i}$$

The form factor $F_{j \to i}$

is the fraction of light leaving dA_j arriving at dA_i taking into account orientation and obstructions

Useful relation

$$A_i F_{i \to j} = A_j F_{j \to i}$$

The equation now becomes

$$B_i = E_i + \rho_i \sum_j F_{i \to j} B_j$$

Rearrange to get

$$B_i - \rho_i \sum_j F_{i \to j} B_j = E_i$$

In matrix form

$$\begin{bmatrix} 1-\rho_1 F_{1\to1} & -\rho_1 F_{1\to2} & \dots & -\rho_1 F_{1\to n} \\ -\rho_2 F_{2\to1} & 1-\rho_2 F_{2\to2} & & -\rho_2 F_{2\to n} \\ & & & \\ -\rho_n F_{n\to1} & -\rho_n F_{n\to2} & & 1-\rho_n F_{n\to n} \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ \\ B_n \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ \\ E_n \end{bmatrix}$$

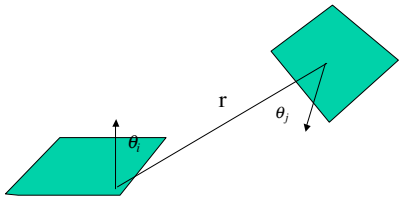
So, in theory, we just compute the Bi's by solving this (large!) matrix equation.

Optional

Optional

The fun part: Computing the $F_{i \to j}$

Without obstruction $dF_{dj \to di} = \frac{\cos\theta_i \cos\theta_j}{\pi r^2} dA_j$



Fancy methods exist for of computing and/or approximating storing form factors (e.g. hemisphere and hemicube methods)

Iterative Solution (gather)

The matrix equation can be solved iteratively (Gauss-Siedel method) starting with a crude estimate of the solution.

More intuitively, consider an estimate $\hat{B}_j$ for the B_j and plug them into our equation to re-estimated them.

$$B_i = E_i + \rho_i \sum_j F_{i \to j} \hat{B}_j$$

This is sometimes referred to as "gather", as we update the brightness of each patch in turn by gathering light from the other patches.

Iterative Solution (cast)

Iteration has the additional benefit that we can provide intermediate, approximate, solutions while the user waits.

But, in the previous version, each patch gets better in turn. Better to have all patches get a little better on each iteration.

This leads to the alternative approach where energy is cast from each patch in turn to update the others with:

$$B_j \quad due \ to \ B_i \ is \ p_j B_i F_{i \to j} \frac{A_i}{A_j} \quad \left(\text{do } \forall B_j \text{ then } \forall B_i\right)$$

A second advantage is that the casting can be done in order of brightness, which is obviously helpful.

Modeling

- Need to usefully represent objects in the world
- Need to provide for easy interaction
  - manual modeling
    - user would like to "fiddle" until it is right (e.g. CAD)
    - user has an idea what an object is like
  - fitting to measurements
    - laser range finder data
- Support rendering/geometric computations

Modeling tools

- Polygon meshes
- Fitting curves to points (from data)
- Fitting curves to points (user interaction)
- Generating shapes with sweeps
- Constructive solid geometry

Polygon Meshes

- Common, straightforward, often built in (e.g. torus mesh)
- Ready to render (many of the representations discussed soon are often be reduced to polygon meshes for rendering)
- Problems
  - Awkward to provide user editing
  - The number of polygons can be very large
    - Some kind of adaptive process makes sense
    - More polygons at high curvature points
    - More polygons where the object is larger
    - Extra care then needs to be taken to avoid temporal aliasing

Explicit curve representation

- Usual representation learned first
- Generally less useful in graphics, but know the term
- Explicit curve is a function of one variable. Examples
  - line, $y=m*x + b$
  - circle (need to glue two together) $y = \pm\text{sqrt}(r*r - x*x)$
- Explicit surface is a function of two variables. Examples
  - plane $z = m*x+ n*y + b$

Implicit representation

- Also less useful for this section, but again, know the term
- An implicit curve is given by the vanishing of some functions
  - circle on the plane (2D), $x*x + y*y - r*r = 0$
  - twisted cubic in space,
    - $x*y - z = 0$, $x*z - y*y=0$, $x*x - y = 0$
- Similarly, an implicit surface is given by the vanishing of some functions
  - sphere in space $x*x + y*y + z*z - r*r = 0$
  - plane $a*x + b*y + c*z + d = 0$

Parametric representation

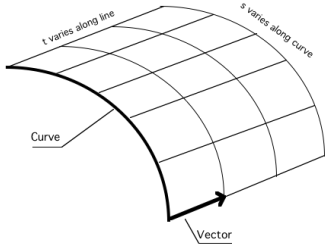
- A parametric **curve** is given as a function of one parameter. Examples:
  - circle as (cos(t), sin(t))
  - twisted cubic as (t, t*t, t*t*t)
- A parametric **surface** is given as a function of two parameters. Examples:
  - sphere as (cos(s) cos(t), sin(s)cos(t), sin(t))
- Advantage - easy to compute normal, easy to render, easy to put patches together, ranges can be easy (e.g. half circle)
- Disadvantage - intersecting with rays for ray tracing can be hard

Generating Surfaces

- We can construct surfaces from curves in a variety of user intuitive ways
  - Extruded surfaces
  - Generalized cones
  - Surfaces of revolution
  - Sweeping (generalized cylinders)
- In many the examples that follow, we will assume that we know how to generate a 3D parameteric curve (studied later)
  - e.g. twisted cubic as (t, t*t, t*t*t)

Extruded surfaces

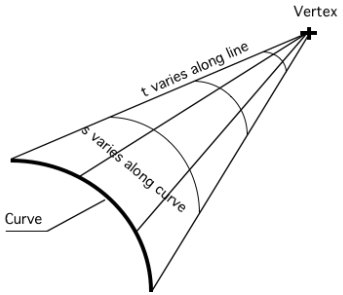
- Geometrical model - Pasta machine
- Take curve and "extrude" surface along vector
- Many human artifacts have this form - rolled steel, etc.



Parametric formula?

Cones

- From every point on a curve, construct a line segment through a single fixed point in space - the vertex
- Curve can be space or plane curve, but shouldn't pass through the vertex

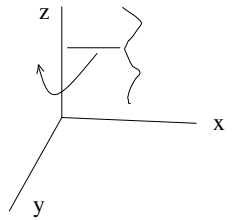


Parametric formula?

Surfaces of revolution

- Plane curve + axis
- "spin" plane curve around axis to get surface
- Choice of plane is arbitrary, choice of axis affects surface
- In the example to the right, curve is on x-z plane, axis is z axis.
- So curve is $(x_c(s), z_c(s))$

Parametric formula?



Sweeps/Generalized Cylinders

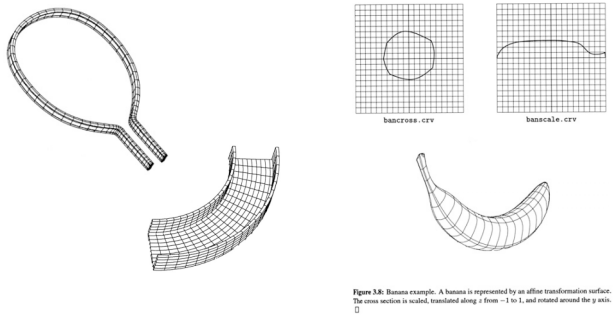
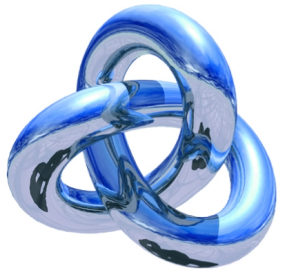


Figure 3.8: Banana example. A banana is represented by an affine transformation surface. The cross section is scaled, translated along z from -1 to 1, and rotated around the y axis.

[Synder 92, via CMU course page]

Sweeps/Generalized Cylinders



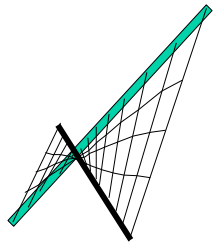
MetaCreations, via CMU course page

Ruled surfaces -1

- Popular, because it's easy to build a curved surface out of straight segments - e.g. pavilions, etc.
- Take two space curves, and join corresponding points—same *s* parameter value—with line segment.
- Even if space curves are lines, the surface is usually curved.

Ruled Surfaces - 2

Easy to explain,
hard to draw!



Ruled surfaces -3

Parameterized form

$$(x(s,t), y(s,t), z(s,t)) = (1-t)(x_1(s), y_1(s), z_1(s)) + t(x_2(s), y_2(s), z_2(s))$$

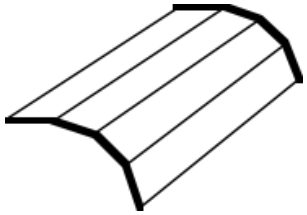
Normals

- Normal is cross product of tangent in t direction and s direction.

$$\left(\frac{\delta x}{\delta t}, \frac{\delta y}{\delta t}, \frac{\delta z}{\delta t}\right) \times \left(\frac{\delta x}{\delta s}, \frac{\delta y}{\delta s}, \frac{\delta z}{\delta s}\right)$$

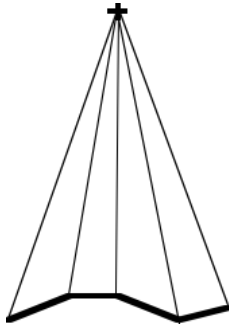
Rendering

- Cylinders: small steps along curve, straight segments along t generate polygons; exact normal is known.



Rendering

- Cone: small steps in s generate straight edges, join with vertex to get triangles, normals known exactly except at vertex.

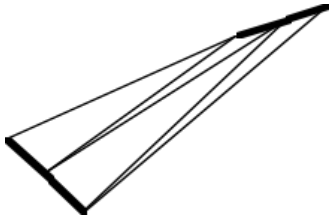


Rendering

- Surface of revolution: small steps in s generate strips, small steps in t along the strip generate edges; join up to form triangles. Normals known exactly.

Rendering

- Ruled surface: steps in s generate polygons, join opposite sides to make triangles - otherwise "non planar polygons" result. Normals known exactly.
- **Must** understand why rectangular sections do not work!



Specifying Curves from Points

- Want to modulate curves via "control" points.
- Strategy depends on application. Possibilities:
  - Force a polynomial of degree N-1 through N points (Lagrange interpolate)
  - Specify a combination of "anchor" points and derivatives (Hermite interpolate)
  - Other "blends" (Bezier, B-splines)--more useful than Lagrange/Hermite

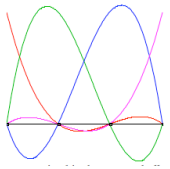
Specifying Curves from Points-II

- Issues:
  - Local versus global control
  - Higher polynomial degree versus stitching lower order polynomials together (stitching-->local control)
  - Continuity of curve and derivatives (geometric, parametric)
  - Polynomials verses other forms
  - Polynomial degree (usually 3--fewer is not flexible enough, and higher gives hard to control wiggles).
  - It is relatively easy to fit a curve through points in explicit form, but we will use parametric form as it more useful in graphics.

Lagrange Interpolate (degree 3)

- Want a parametric curve that passes through (interpolates) four points.
- Use the points to combine four Lagrange polynomials (blending functions)
- As the parameter goes through each of 4 particular values, one blending function is 1, and the other 3 are zero.

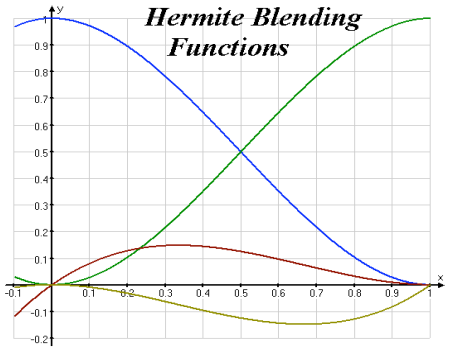
$$\sum_{i \in \text{points}} p_i \phi_i^{(l)}(t)$$



Hermite (H&B, page 426)

- Hermite interpolate
  - Curve passes through specified points **and** has specified derivatives at those points.
- Standard degree 3 case: 2 points, 2 derrivatives at those points
- 4 functions of degree 3, two each of two kinds
  - one at an endpoint, zero at the other, AND derrivative is zero at both
  - derivative is one at an endpoint and zero at others, AND value is zero at the endpoints.

$$\sum_{i \in \text{points}} p_i \phi_i^{(h)}(t) + \sum_{i \in \text{points}} v_i \phi_i^{(hd)}(t)$$



From www.cs.virginia.edu/~gfx/Courses/ 2002/Intro.fall.02

Blended curves

- Assume degree 3
- Includes Hermite, Bézier and others

$$Q(t) = \begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix} = C * T(t) = \begin{bmatrix} a_x & b_x & c_x & d_x \\ a_y & b_y & c_y & d_y \\ a_z & b_z & c_z & d_z \end{bmatrix} \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix}$$

Blended curves

- General pattern: Decompose the matrix C into two factors
  - One factor encodes the "control" or data points
  - The second factor is the blending functions

$$Q(t) = \begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix} = [G_1 \quad G_2 \quad G_3 \quad G_4] \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{bmatrix} \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix}$$

Hermite

- Geometry matrix
  - First two columns are endpoints
  - Next two columns are derivatives at those points
- Blending function matrix

$$M_H = \begin{bmatrix} 2 & -3 & 0 & 1 \\ -2 & 3 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 1 & -1 & 0 & 0 \end{bmatrix}$$

Where does this come from?

Hermite

$$Q(t) = \begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix} = [G_1 \quad G_2 \quad G_3 \quad G_4] M_H \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix}$$

Can solve for M_H using x(t) only (or y(t), or z(t))

$$x(t) = [G_{1x} \quad G_{2x} \quad G_{3x} \quad G_{4x}] * \underbrace{M_H * T(t)}$$

Vector of blending functions; compare with previous representation.

$$\sum_{i \in \text{points}} p_i \phi_i^{(h)}(t) + \sum_{i \in \text{points}} v_i \phi_i^{(hd)}(t)$$

Hermite

$$x(t) = [G_{1x} \quad G_{2x} \quad G_{3x} \quad G_{4x}] \; M_H * \mathbf{T}(t)$$

We have

$x(0) = G_{1x}$ so $M_H * T(0) = ?$
$x(1) = G_{2x}$ so $M_H * T(1) = ?$
$x'(0) = G_{3x}$ so $M_H * T'(0) = ?$
$x'(1) = G_{4x}$ so $M_H * T'(1) = ?$

Hermite

$x(0) = G_{1x}$ so $M_H * \mathbf{T}(0) = [1\ 0\ 0\ 0]^T$
$x(1) = G_{2x}$ so $M_H * \mathbf{T}(1) = [0\ 1\ 0\ 0]^T$
$x'(0) = G_{3x}$ so $M_H * \mathbf{T}'(0) = [0\ 0\ 1\ 0]^T$
$x'(1) = G_{4x}$ so $M_H * \mathbf{T}'(1) = [0\ 0\ 0\ 1]^T$

So

$$M_H * [\mathbf{T(0)} \quad \mathbf{T(1)} \quad \mathbf{T'(0)} \quad \mathbf{T'(1)}] = I$$

So

$$M_H = [\mathbf{T(0)} \quad \mathbf{T(1)} \quad \mathbf{T'(0)} \quad \mathbf{T'(1)}]^{-1}$$

Hermite

$$\mathbf{T}(t) = \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix} \qquad \mathbf{T}'(t) = ?$$

Hermite

$$\mathbf{T}(t) = \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix} \qquad \mathbf{T}'(t) = \begin{bmatrix} 3t^2 \\ 2t^2 \\ 1 \\ 0 \end{bmatrix}$$

$$\mathbf{T(0)} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \quad \mathbf{T(1)} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \quad \mathbf{T'(0)} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad \mathbf{T'(1)} = \begin{bmatrix} 3 \\ 2 \\ 1 \\ 0 \end{bmatrix}$$

Hermite

Recall that

$$M_H * [\mathbf{T(0)} \quad \mathbf{T(1)} \quad \mathbf{T'(0)} \quad \mathbf{T'(1)}] = I$$

And thus we seek

$$M_H = [\mathbf{T(0)} \quad \mathbf{T(1)} \quad \mathbf{T'(0)} \quad \mathbf{T'(1)}]^{-1}$$

$$M_H = \begin{vmatrix} 0 & 1 & 0 & 3 \\ 0 & 1 & 0 & 2 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{vmatrix}^{-1}$$

Hermite

Finally

$$M_H = \begin{vmatrix} 0 & 1 & 0 & 3 \\ 0 & 1 & 0 & 2 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{vmatrix}^{-1} = \begin{bmatrix} 2 & -3 & 0 & 1 \\ -2 & 3 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 1 & -1 & 0 & 0 \end{bmatrix}$$