The 3D `Asymptote` Generalization of the `MetaPost` Bezier Interpolation Algorithms

John Bowman

University of Alberta

July 16, 2007

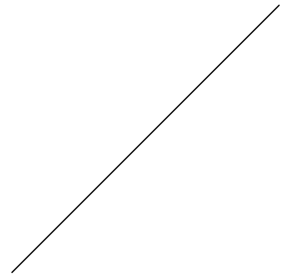
`http://www.math.ualberta.ca/~bowman`

History

- TeX and METAFONT (Knuth, 1979)
- MetaPost (Hobby, 1989): 2D Bezier Control Point Selection
- Asymptote (Hammerlindl, Bowman, Prince, 2004): 2D & 3D

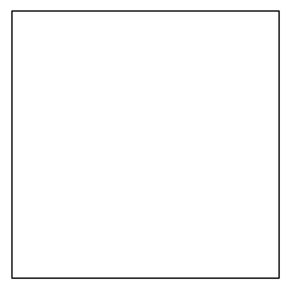
Cartesian Coordinates

```
draw((0,0)--(100,100));
```



- units are `PostScript` *big points* (1 `bp` = 1/72 `inch`)
- `--` means join the points with a linear segment to create a *path*
- cyclic path:

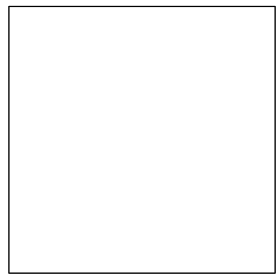
```
draw((0,0)--(100,0)--(100,100)--(0,100)--cycle);
```



Scaling to a Given Size

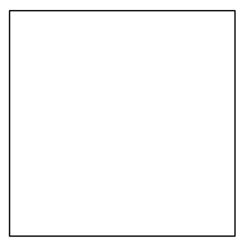
- `PostScript` units are often inconvenient.
- Instead, scale user coordinates to a specified final size:

```
size(101,101);
draw((0,0)--(1,0)--(1,1)--(0,1)--cycle);
```



- One can also specify the size in `cm`:

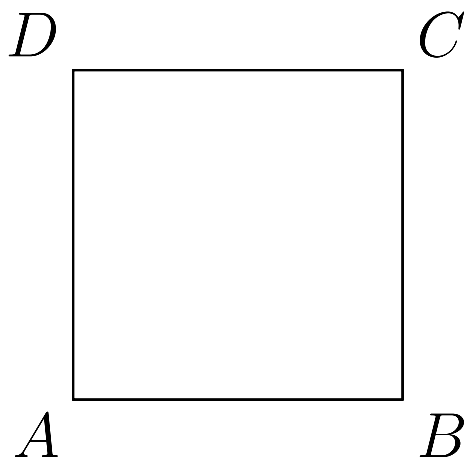
```
size(0,3cm);
draw(unitsquare);
```



Labels

- Adding and aligning $\LaTeX$ labels is easy:

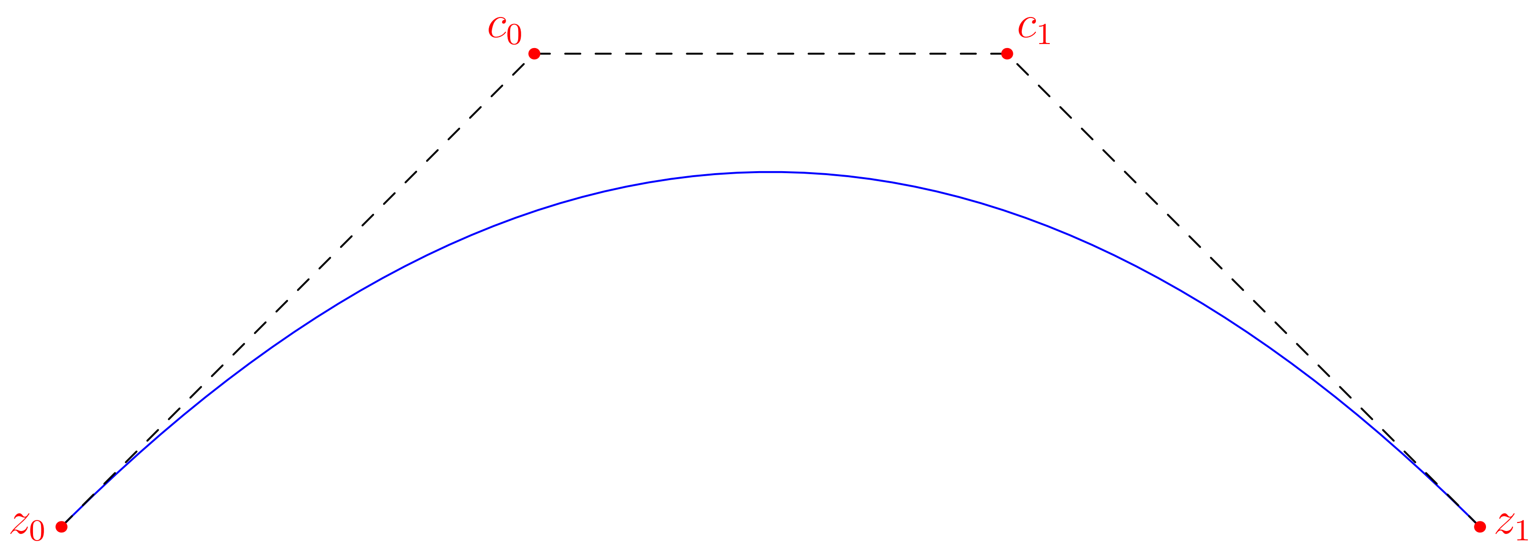
```
size(0,3cm);
draw(unitsquare);
label("$A$",(0,0),SW);
label("$B$",(1,0),SE);
label("$C$",(1,1),NE);
label("$D$",(0,1),NW);
```



2D Bezier Splines

- Using `..` instead of `--` specifies a *Bezier cubic spline*:

```
draw(z0 .. controls c0 and c1 .. z1,blue);
```

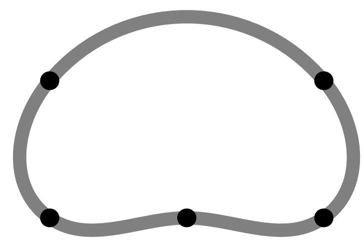


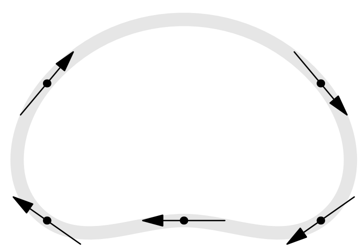
$$(1-t)^3 z_0 + 3t(1-t)^2 c_0 + 3t^2(1-t) c_1 + t^3 z_1, \qquad t \in [0,1].$$

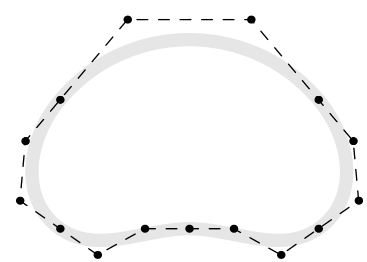
Smooth Paths

- Asymptote can choose control points for you, using the algorithms of Hobby [1986] and Knuth [1986]:

```
pair[] z={(0,0), (0,1), (2,1), (2,0), (1,0)};

draw(z[0]..z[1]..z[2]..z[3]..z[4]..cycle,
     grey+linewidth(5));
dot(z,linewidth(7));
```



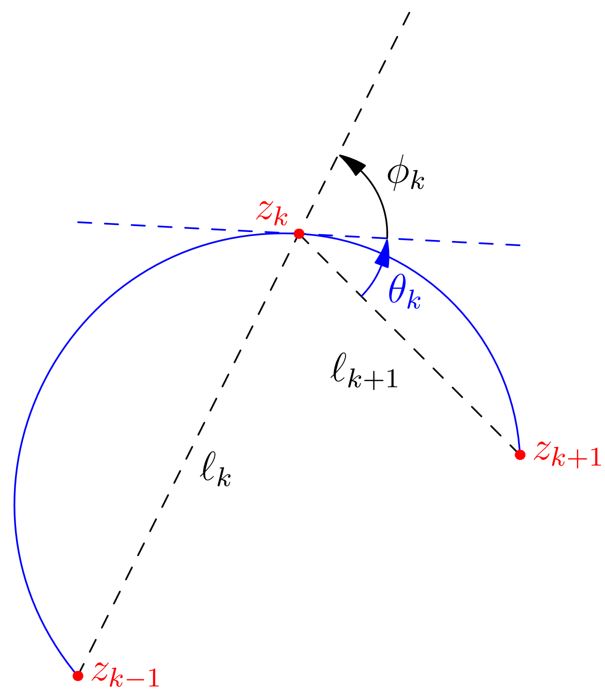




Hobby's 2D Direction Algorithm

- A tridiagonal system of linear equations is solved to determine any unspecified directions θ_k and ϕ_k through each knot z_k:

$$\frac{\theta_{k-1} - 2\phi_k}{\ell_k} = \frac{\phi_{k+1} - 2\theta_k}{\ell_{k+1}}.$$



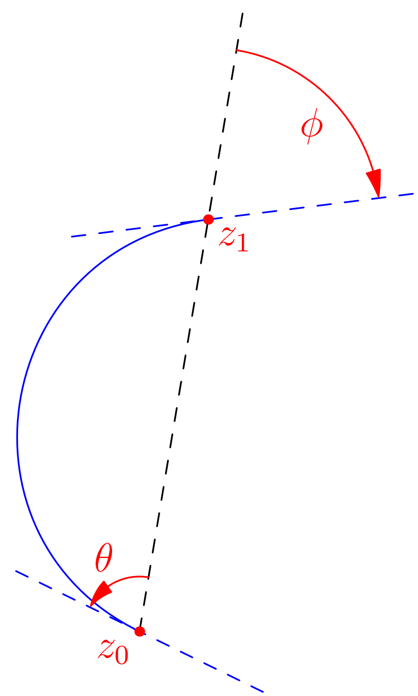
- The resulting shape may be adjusted by modifying optional *tension* parameters and *curl* boundary conditions.

Hobby's 2D Control Point Algorithm

- Having prescribed outgoing and incoming path directions $e^{i\theta_0}$ at node z_0 and $e^{i\theta_1}$ at node z_1 relative to the vector $z_1 - z_0$, the control points are determined as:

$$u = z_0 + e^{i\theta}(z_1 - z_0) f(\theta, -\phi),$$
$$v = z_1 - e^{i\phi}(z_1 - z_0) f(-\phi, \theta),$$

where the relative distance function $f(\theta, \phi)$ is given by Hobby [1986].



Bezier Curves in 3D

- Apply an affine transformation

$$x'_i = A_{ij} x_j + C_i$$

to a Bezier curve:

$$x(t) = \sum_{k=0}^{3} B_k(t) P_k, \qquad t \in [0, 1].$$

$$x'_i = A_{ij} x_j + C_i.$$

- The resulting curve is also a Bezier curve:

$$\begin{aligned} x'_i(t) &= \sum_{k=0}^{3} B_k(t) A_{ij} (P_k)_j + C_i \\ &= \sum_{k=0}^{3} B_k(t) P'_k, \end{aligned}$$

where P'_k is the transformed k^{th} control point, noting $\sum_{k=0}^{3} B_k(t) = 1$.

3D Generalization of Hobby's algorithm

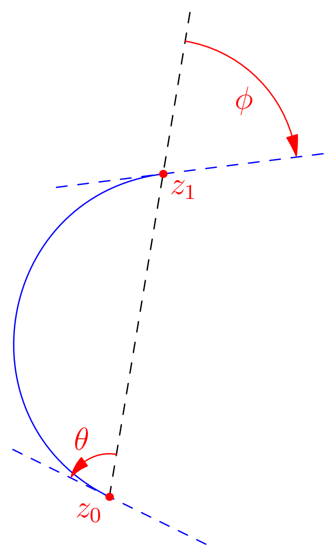
- Must reduce to 2D algorithm in planar case.
- Determine directions by applying Hobby's algorithm in the plane containing z_{k-1}, z_k, z_{k+1}.
- The only ambiguity that can arise is the overall sign of the angles, which relates to viewing each 2D plane from opposing normal directions.
- A reference vector based on the mean unit normal of successive segments can be used to resolve such ambiguities.

3D Control Point Algorithm

- Hobby's control point algorithm can be generalized to 3D by expressing it in terms of the absolute directions ω_0 and ω_1:

$$u = z_0 + \omega_0 \left| z_1 - z_0 \right| f(\theta, -\phi),$$

$$v = z_1 - \omega_1 \left| z_1 - z_0 \right| f(-\phi, \theta),$$

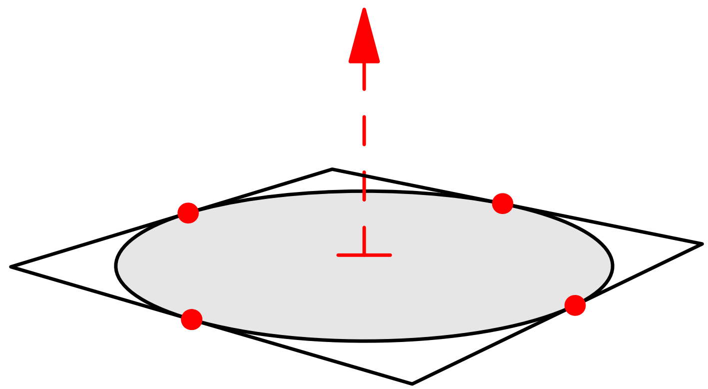


interpreting θ and ϕ as the angle between the corresponding path direction vector and $z_1 - z_0$.

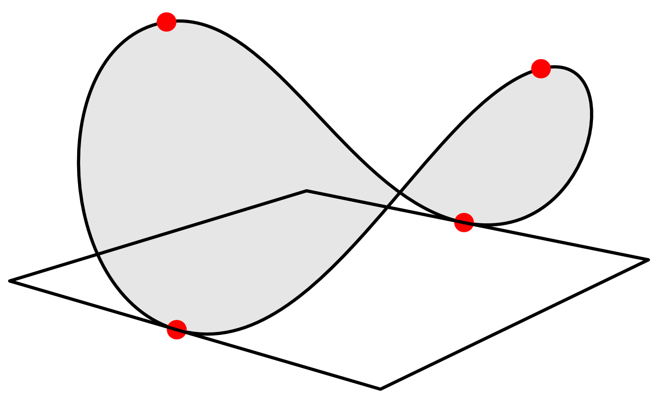
- In this case there is an unambiguous reference vector for determining the relative sign of the angles ϕ and θ.

3D saddle example

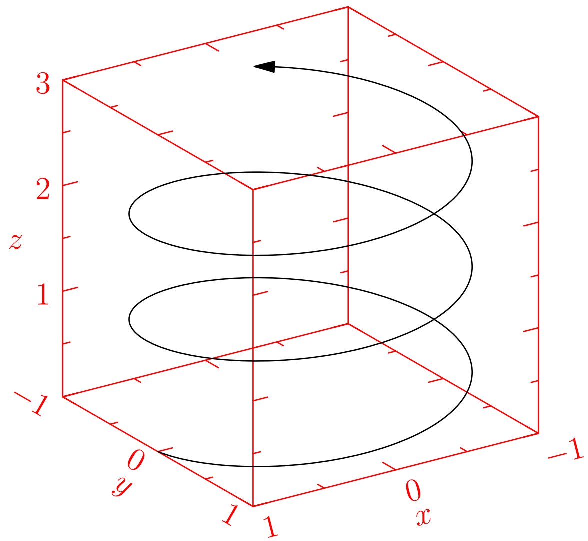
- A unit circle in the X–Y plane may be filled and drawn with: (1,0,0)..(0,1,0)..(-1,0,0)..(0,-1,0)..cycle

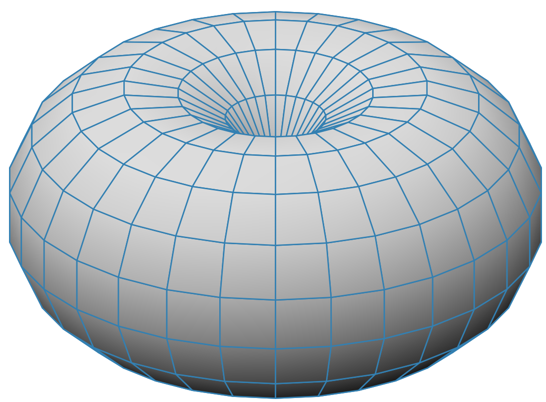


and then distorted into a saddle:
(1,0,0)..(0,1,1)..(-1,0,0)..(0,-1,1)..cycle



3D graphs and surfaces





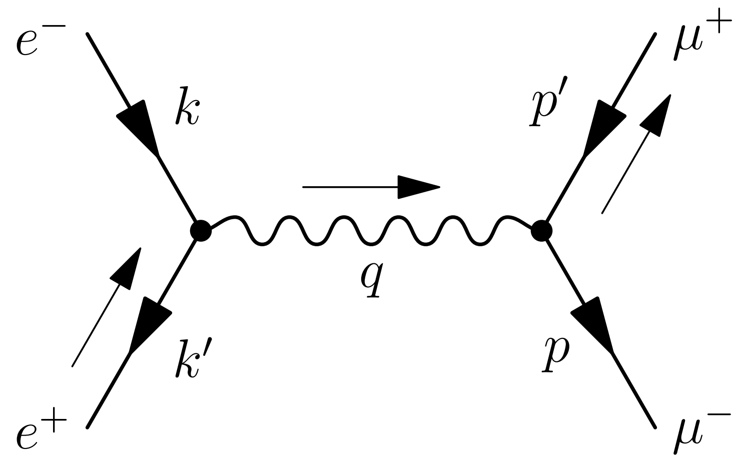
Affine Transforms

- Affine transformations can be applied to pairs, triples, paths, pens, and even whole pictures:

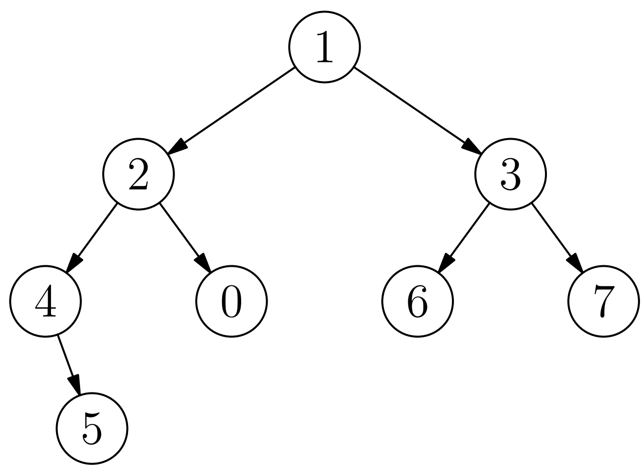
```
transform t=rotate(90);
write(t*(1,0));  // Writes (0,1).

fill(P,blue);
fill(shift(2,0)*reflect((0,0),(0,1))*P, red);
fill(shift(4,0)*rotate(30)*P, yellow);
fill(shift(6,0)*yscale(0.7)*xscale(2)*P, green);
```

Packages

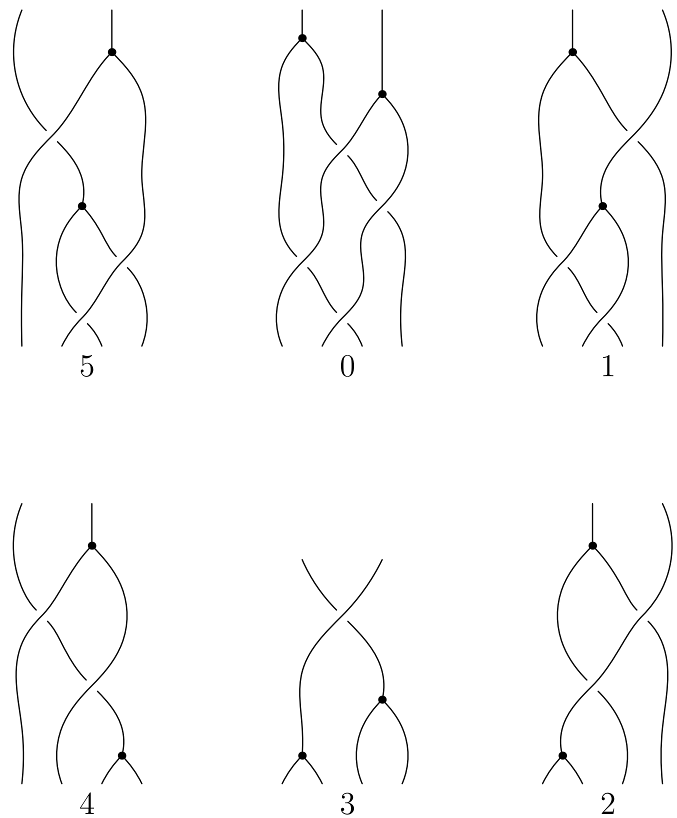
- There are packages for Feynman diagrams,



data structures,

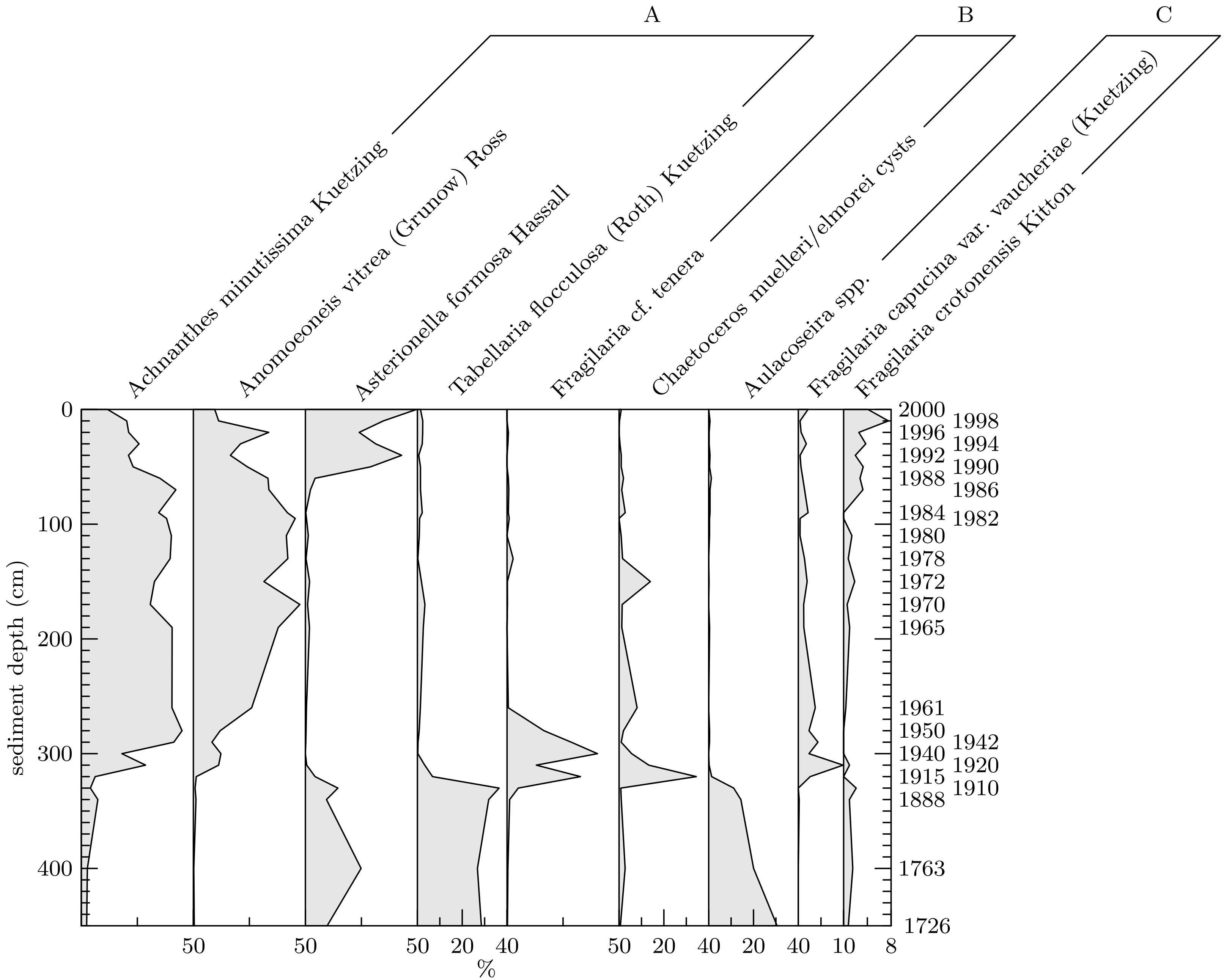


algebraic knot theory:



$$\begin{aligned}\Phi\Phi(x_1,x_2,x_3,x_4,x_5) = \quad & \rho_{4b}(x_1+x_4,x_2,x_3,x_5)+\rho_{4b}(x_1,x_2,x_3,x_4) \\ & +\rho_{4a}(x_1,x_2+x_3,x_4,x_5)-\rho_{4b}(x_1,x_2,x_3,x_4+x_5) \\ - \quad & \rho_{4a}(x_1+x_2,x_3,x_4,x_5)-\rho_{4a}(x_1,x_2,x_4,x_5).\end{aligned}$$

Scientific Graphs



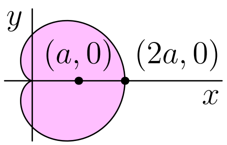
Slide Presentations

- Asymptote has a package for preparing slides.
- It even supports embedded hi-resolution PDF movies.

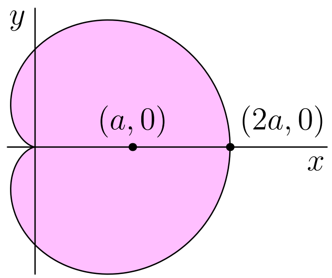
```
title("Slide Presentations");
item("Asymptote has a package for preparing slides.");
item("It even supports embedded hi-resolution PDF movies.");
  ...
```

Automatic Sizing

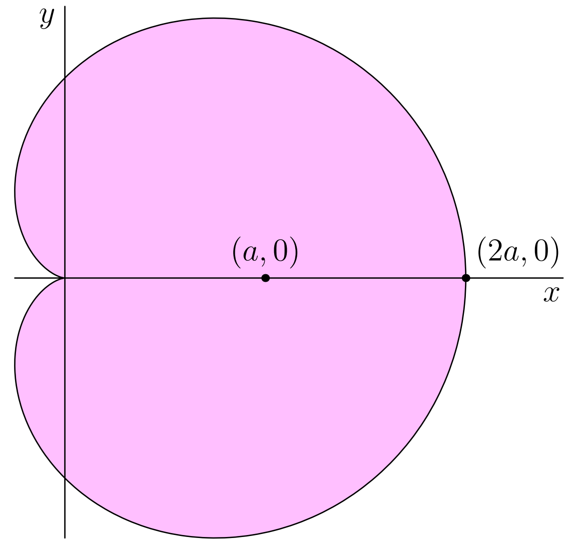
- Figures can be specified in user coordinates, then automatically scaled to the final size.



`size(0,50);`



`size(0,100);`



`size(0,200);`

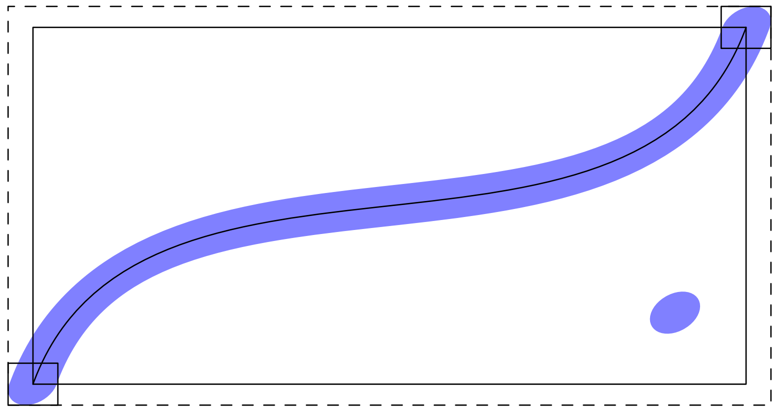
Deferred Drawing

- We can't draw a graphical object until we know the scaling factors for the user coordinates.
- Instead, store a function that when given the scaling information, draws the scaled object.

```
void draw(picture pic=currentpicture, path g, pen p=currentpen) {
  pic.add(new void(frame f, transform t) {
      draw(f,t*g,p);
    });
  pic.addPoint(min(g),min(p));
  pic.addPoint(max(g),max(p));
}
```

Coordinates

- Store bounding box information as a sum of user and true-size coordinates:



```
pic.addPoint(min(g),min(p));
pic.addPoint(max(g),max(p));
```

- Filling ignores the pen width:

```
pic.addPoint(min(g),(0,0));
pic.addPoint(max(g),(0,0));
```

- Communicate with LaTeX to determine label sizes:

$$\boxed{E=mc^2}$$

Sizing

- When scaling the final figure to a given size S, we first need to determine a scaling factor $a > 0$ and a shift b so that all of the coordinates when transformed will lie in the interval $[0, S]$. That is, if u and t are the user and truesize components:

$$0 \leq au + t + b \leq S.$$

- We are maximizing the variable a subject to a number of inequalities. This is a linear programming problem that can be solved by the simplex method.

Sizing

- Every addition of a coordinate (t, u) adds two restrictions

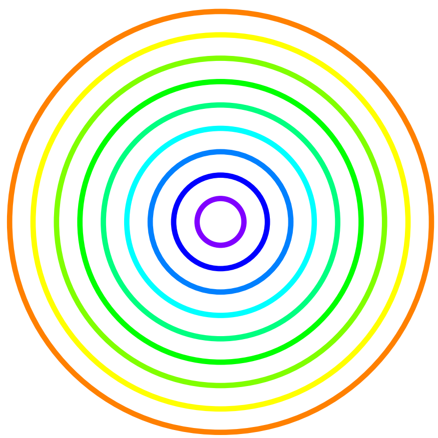
$$au + t + b \geq 0,$$

$$au + t + b \leq S,$$

and each drawing component adds two coordinates.

- A figure could easily produce thousands of restrictions, making the simplex method impractical.

- Most of these restrictions are redundent, however. For instance, with concentric circles, only the largest circle needs to be accounted for.



Redundant Restrictions

- In general, if $u \le u'$ and $t \le t'$ then
$$au + t + b \le au' + t' + b$$
for all choices of $a > 0$ and b, so
$$0 \le au + t + b \le au' + t' + b \le S.$$

- This defines a partial ordering on coordinates. When sizing a picture, the program first computes which coordinates are maximal (or minimal) and only sends effective restraints to the simplex algorithm.

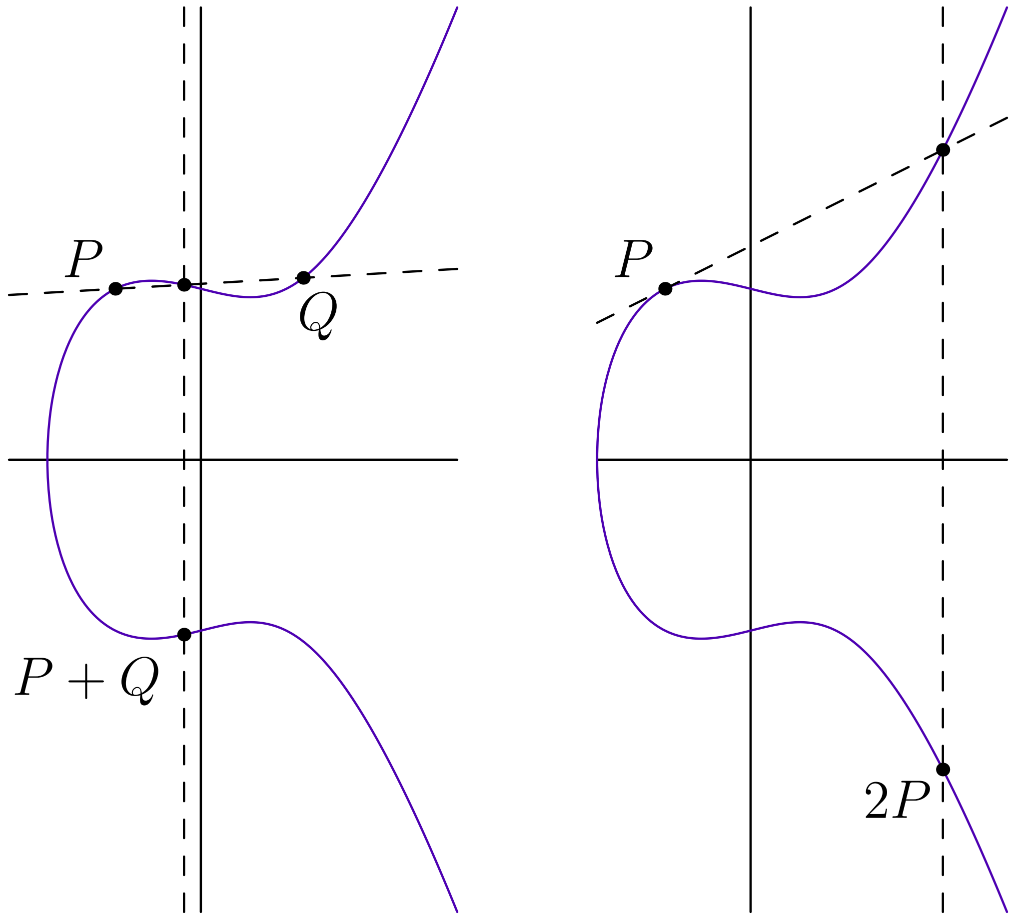
- In practice, the linear programming problem will have less than a dozen restraints.

- All picture sizing is implemented in Asymptote code.

Infinite Lines

- Deferred drawing allows us to draw infinite lines.

```
drawline(P, Q);
```



References

[Hobby 1986] J. D. Hobby, Discrete Comput. Geom., **1**:123, 1986.

[Knuth 1986] D. E. Knuth, *The METAFONTbook*, Addison-Wesley, Reading, Massachusetts, 1986.

Asymptote: The Vector Graphics Language

`http://asymptote.sf.net`

(freely available under the GNU public license)