Interactive TeX-Aware 3D Vector Graphics

John Bowman and Andy Hammerlindl

Department of Mathematical and Statistical Sciences
University of Alberta

Collaborators: Orest Shardt, Michail Vidiassov

June 30, 2010

`http://asymptote.sf.net/intro.pdf`

History

- 1979: TeX and METAFONT (Knuth)

- 1986: 2D Bézier control point selection (Hobby)

- 1989: MetaPost (Hobby)

- 2004: Asymptote

  – 2004: initial public release (Hammerlindl, Bowman, & Prince)

  – 2005: 3D Bézier control point selection (Bowman)

  – 2008: 3D interactive TeX within PDF files (Shardt & Bowman)

  – 2009: 3D billboard labels that always face camera (Bowman)

  – 2010: 3D PDF enhancements (Vidiassov & Bowman)

Statistics (as of June, 2010)

- Runs under Linux/UNIX, Mac OS X, Microsoft Windows.

- 4000 downloads/month from primary
  `asymptote.sourceforge.net` site alone.

- 80 000 lines of low-level C++ code.

- 36 000 lines of high-level Asymptote code.

Cartesian Coordinates

- Asymptote's graphical capabilities are based on four primitive commands: `draw`, `label`, `fill`, `clip` [BH08]

  ```
  draw((0,0)--(100,100));
  ```
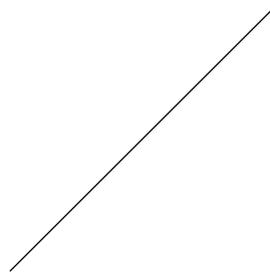
- units are `PostScript` *big points* $(1 \text{ bp} = 1/72 \text{ inch})$

- `--` means join the points with a linear segment to create a *path*

- *cyclic* path:

  ```
  draw((0,0)--(100,0)--(100,100)--(0,100)--cycle);
  ```
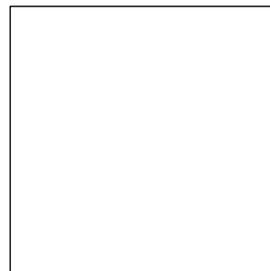
Scaling to a Given Size

- **PostScript** units are often inconvenient.

- Instead, scale user coordinates to a specified final size:

```
size(100,100);
draw((0,0)--(1,0)--(1,1)--(0,1)--cycle);
```
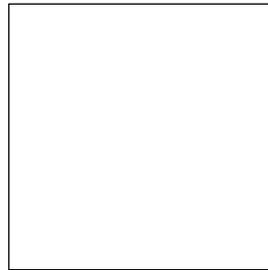
- One can also specify the size in **cm**:

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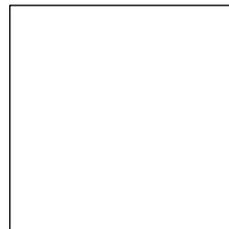
Labels

- Adding and aligning LaTeX labels is easy:

```
size(6cm);
draw(unitsquare);
label("$A$",(0,0),SW);
label("$B$",(1,0),SE);
label("$C$",(1,1),NE);
label("$D$",(0,1),NW);
```
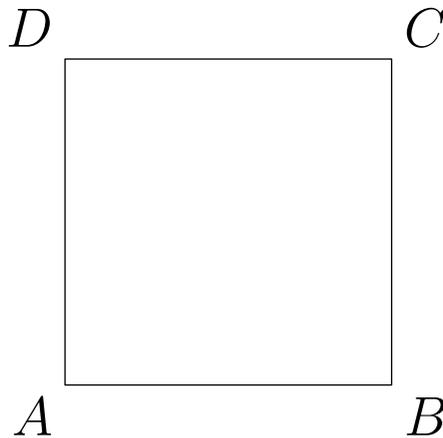
2D Bézier Splines

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

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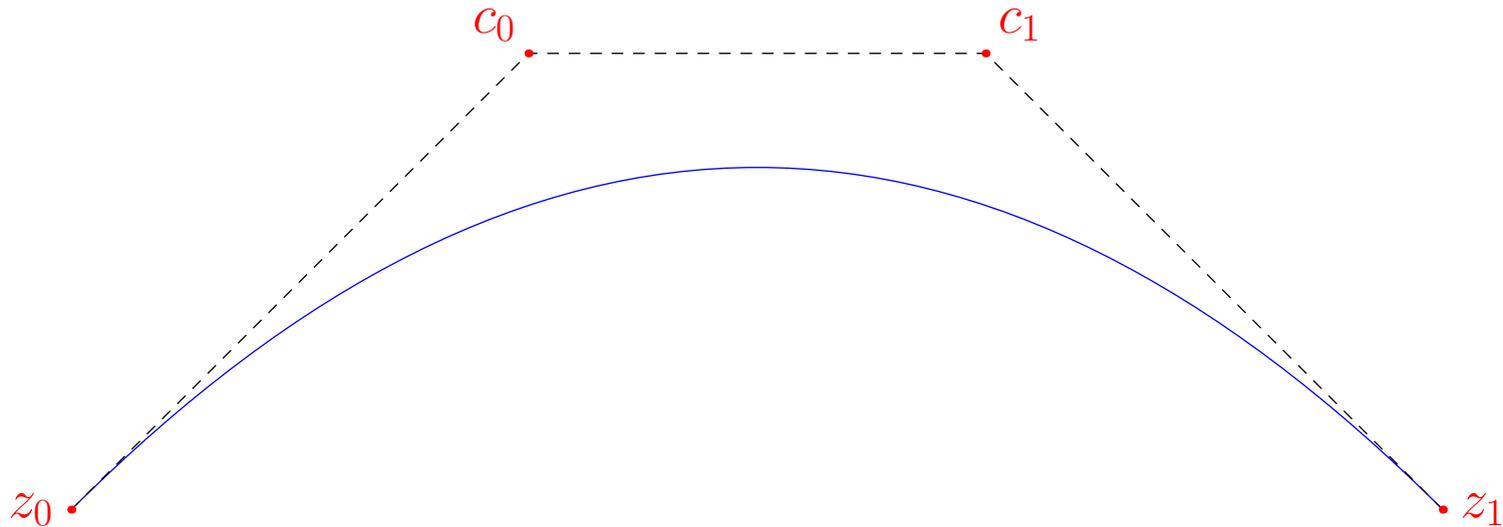

$$(1-t)^3 z_0 + 3t(1-t)^2 c_0 + 3t^2(1-t)c_1 + t^3 z_1, \qquad t \in [0,1].$$

Smooth Paths

- Asymptote can choose control points for you, using the algorithms of Hobby and Knuth [Hob86, Knu86]:

```
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));
```
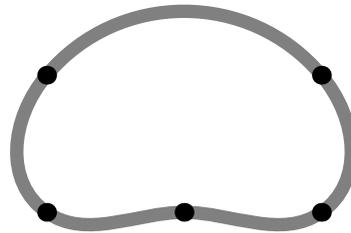
- First, linear equations involving the curvature are solved to find the direction through each knot. Then, control points along those directions are chosen:
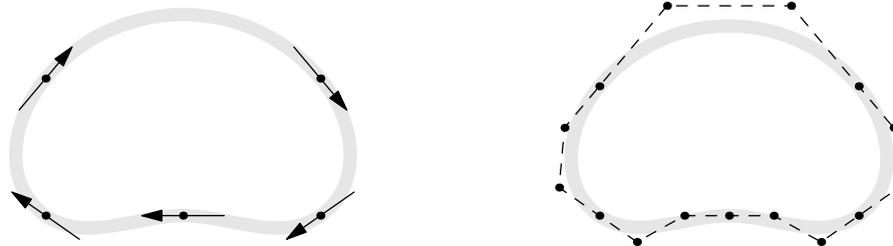
Filling

- The `fill` primitive to fill the inside of a path:

```
path star;
for(int i=0; i < 5; ++i)
  star=star--dir(90+144i);
star=star--cycle;

fill(star,orange+zerowinding);
draw(star,linewidth(3));

fill(shift(2,0)*star,blue+evenodd);
draw(shift(2,0)*star,linewidth(3));
```
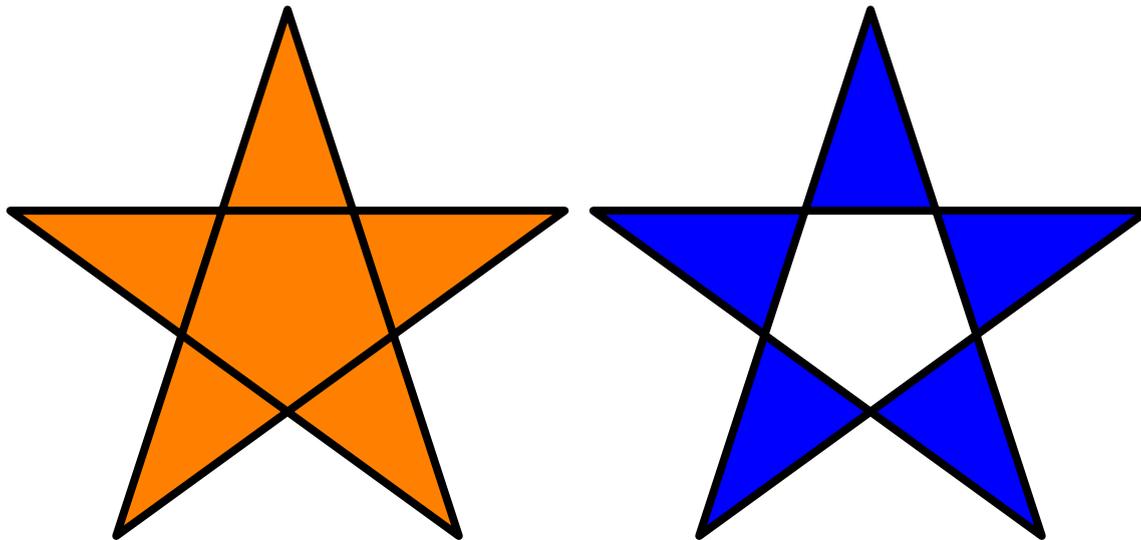
# Filling

- Use a list of paths to fill a region with holes:

```
path[] p={scale(2)*unitcircle, reverse(unitcircle)};
fill(p,green+zerowinding);
```
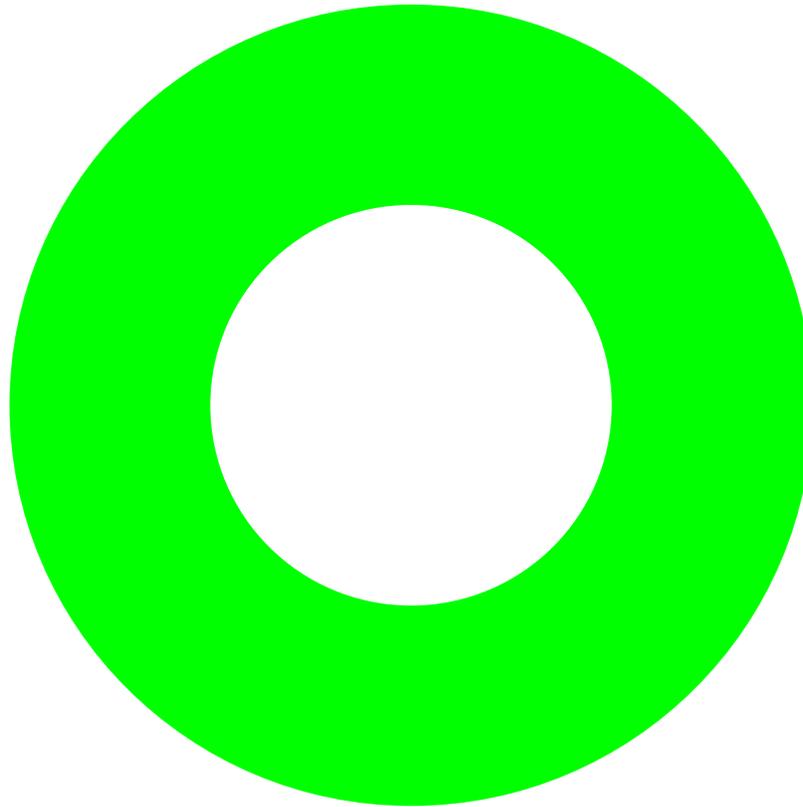
Clipping

- Pictures can be clipped to a path:

```
fill(star,orange+zerowinding);
clip(scale(0.7)*unitcircle);
draw(scale(0.7)*unitcircle);
```
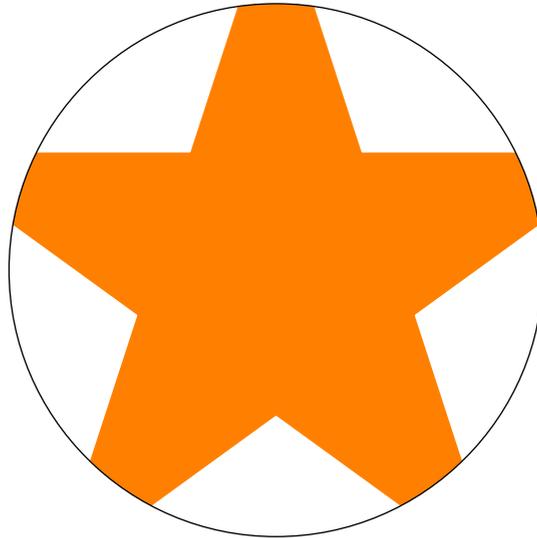
Affine Transforms

- Affine transformations: shifts, rotations, reflections, and scalings can be applied to pairs, paths, pens, strings, and even whole pictures:

```
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);
```

Modules

- There are modules for Feynman diagrams,
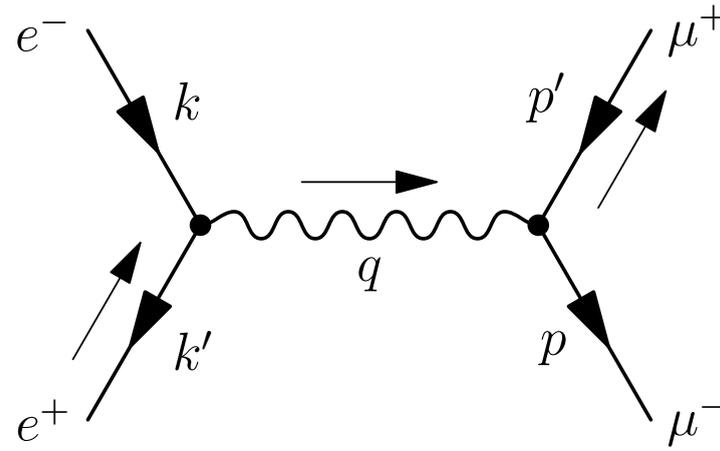
data structures,
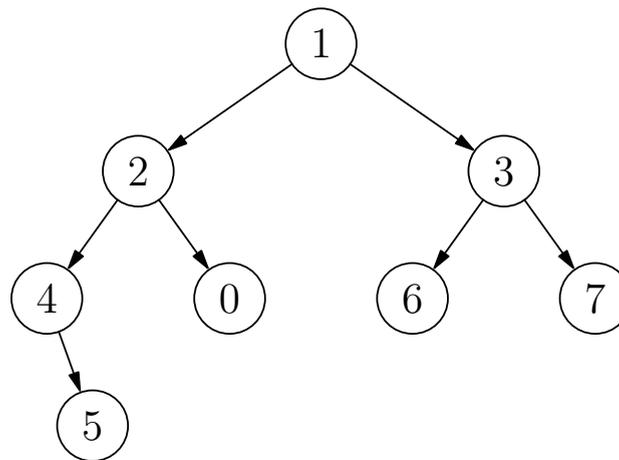
algebraic knot theory:
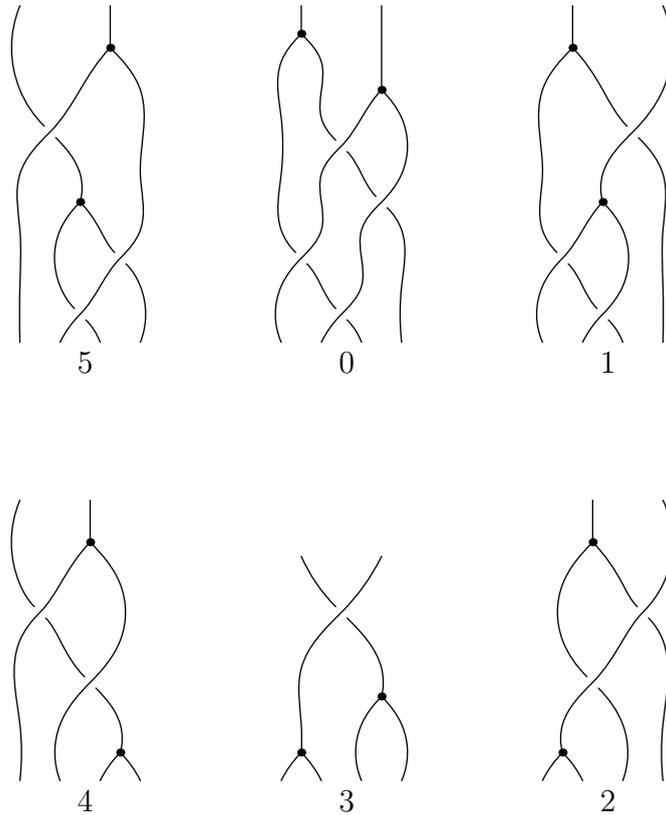
$$\Phi\Phi(x_1, x_2, x_3, x_4, x_5) = \quad \rho_{4b}(x_1 + x_4, x_2, x_3, x_5) + \rho_{4b}(x_1, x_2, x_3, x_4)$$
$$+ \quad \rho_{4a}(x_1, x_2 + x_3, x_4, x_5) - \rho_{4b}(x_1, x_2, x_3, x_4 + x_5)$$
$$- \quad \rho_{4a}(x_1 + x_2, x_3, x_4, x_5) - \rho_{4a}(x_1, x_2, x_4, x_5).$$

Scientific Graph

Proportion of crows
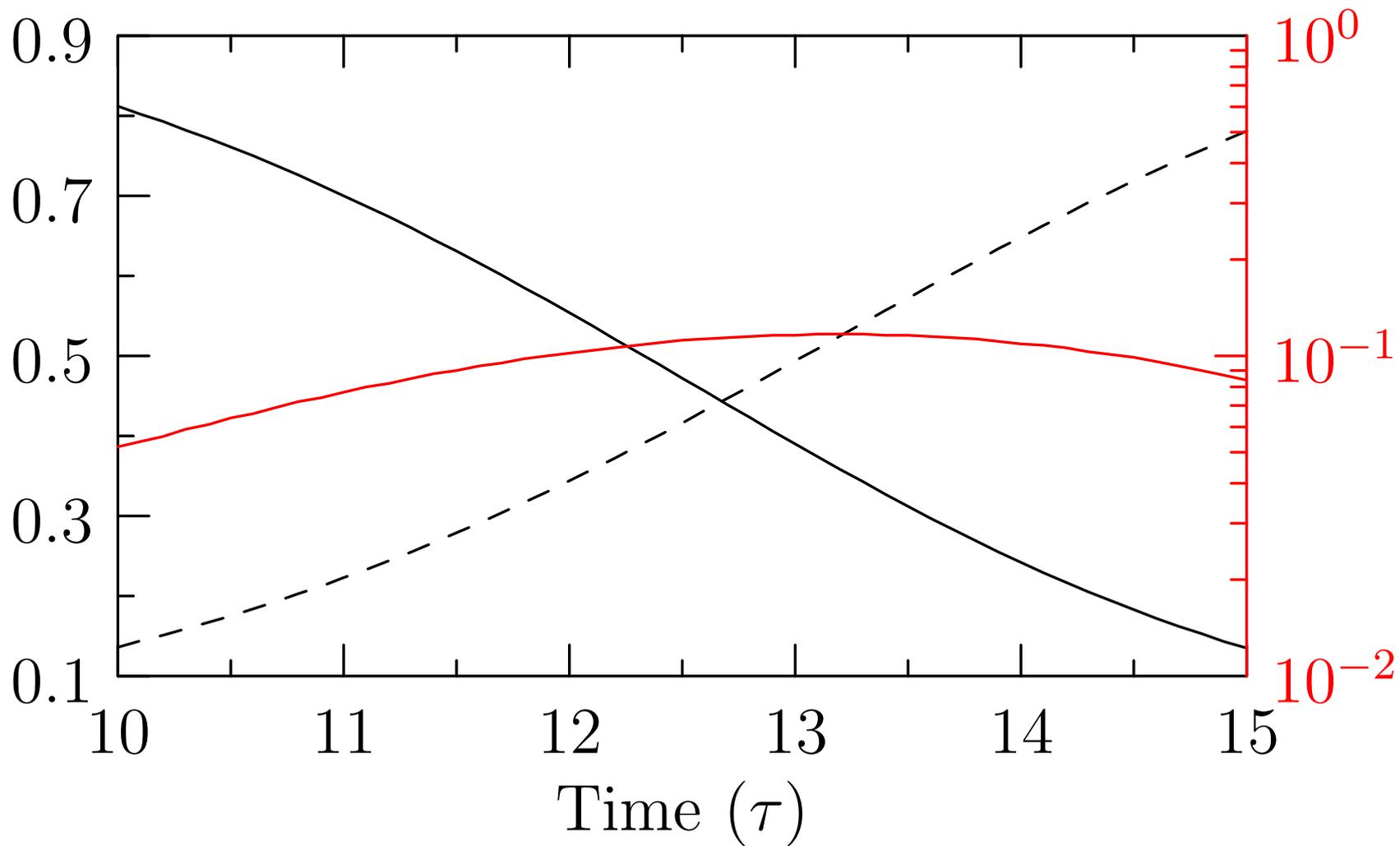
Images and Contours

$f(x, y)$
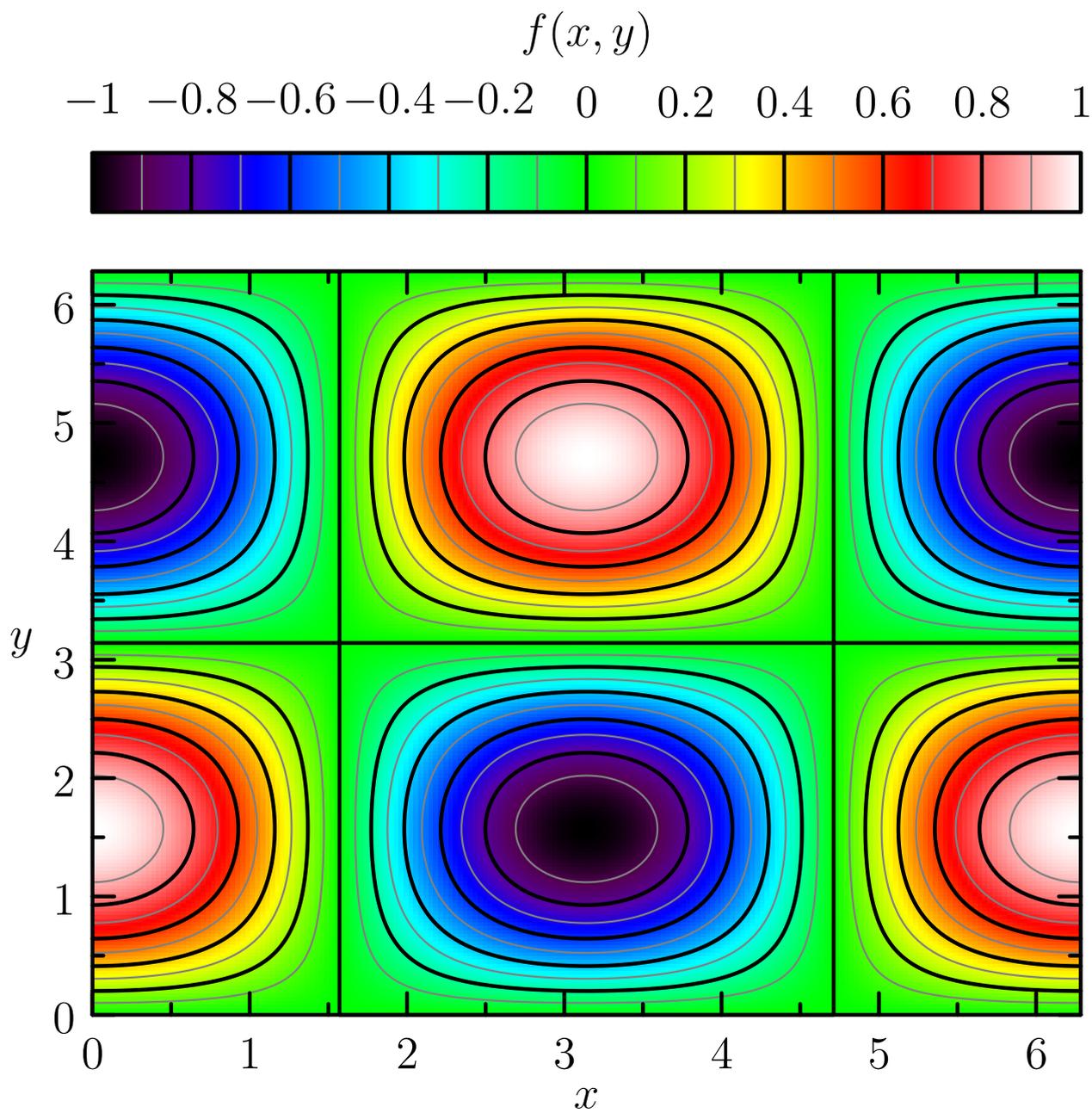
# Multiple Graphs
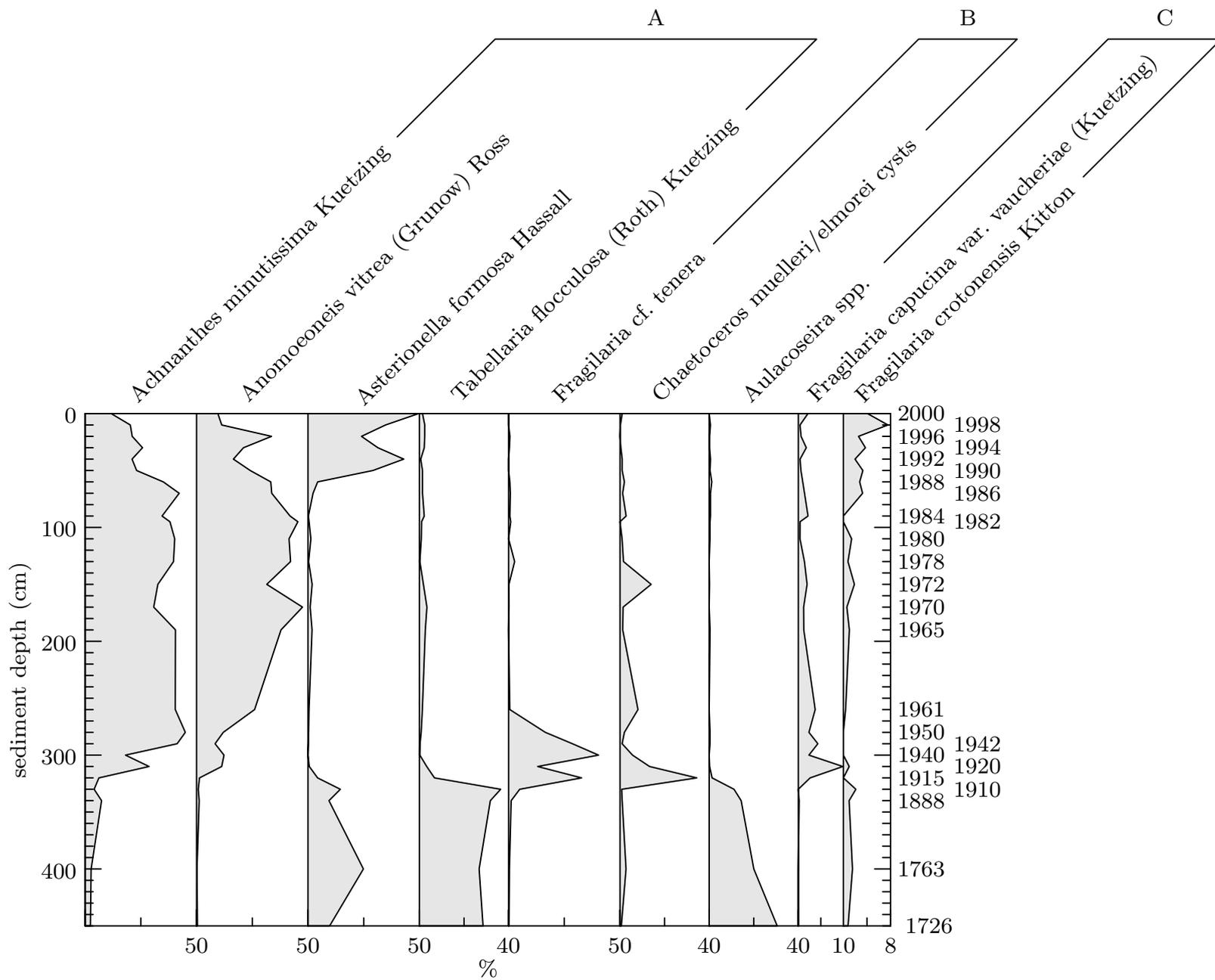
Hobby's 2D Direction Algorithm

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

$$\frac{\theta_{k-1} - 2\phi_k}{\ell_k} = \frac{\phi_{k+1} - 2\theta_k}{\ell_{k+1}}.$$
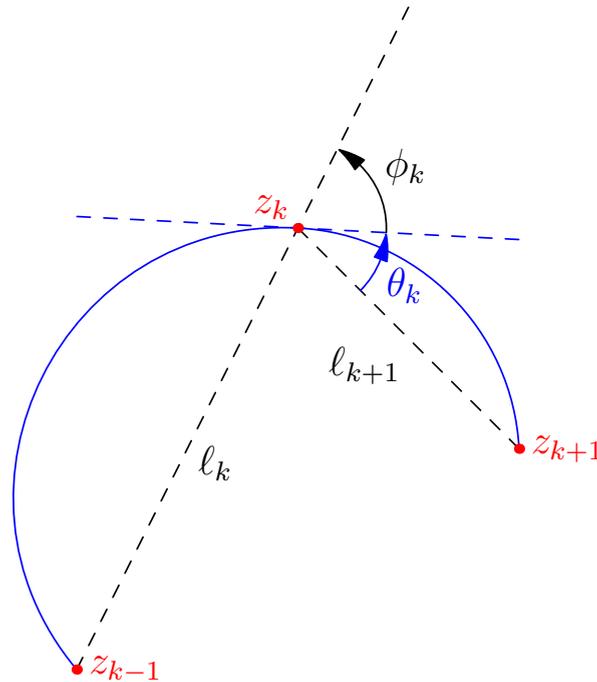
- The resulting shape may be adjusted by modifying optional *tension* parameters and *curl* boundary conditions.

Hobby's 2D Control Point Algorithm

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

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

where the relative distance function $f(\theta, \phi)$ is given by Hobby [1986].
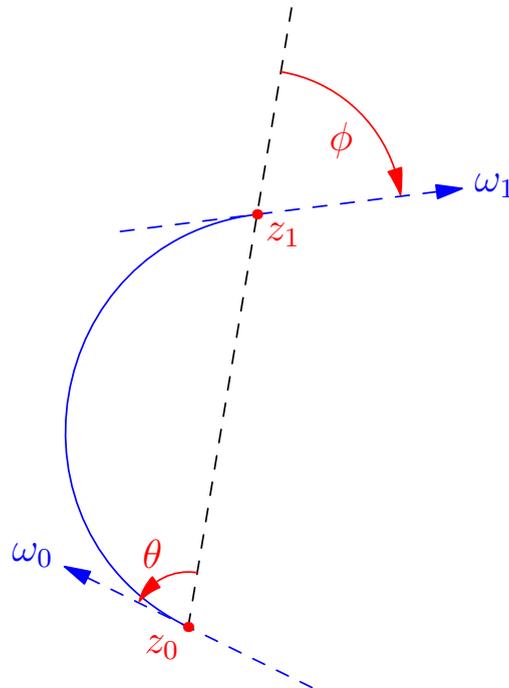
3D Generalization of Direction Algorithm

- 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 [Bow07, BS09]

3D Control Point Algorithm

- Express Hobby's algorithm in terms of the absolute directions ω_0 and ω_1:

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

$$v = z_1 - \omega_1 \left| z_1 - z_0 \right| f(-\phi, \theta),$$
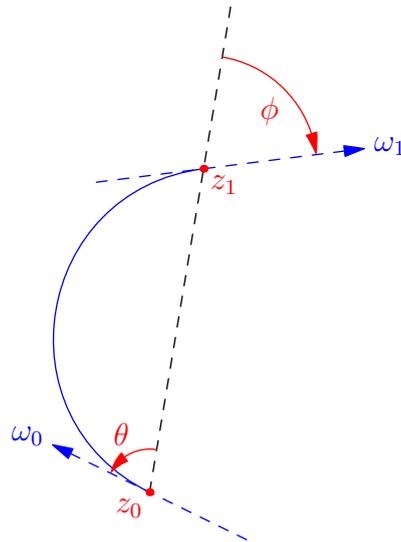
interpreting θ and ϕ as the angle between the corresponding path direction vector and $z_1 - z_0$.

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

Interactive 3D Saddle

- A unit circle in the X–Y plane may be constructed with:
  `(1,0,0)..(0,1,0)..(-1,0,0)..(0,-1,0)..cycle`:
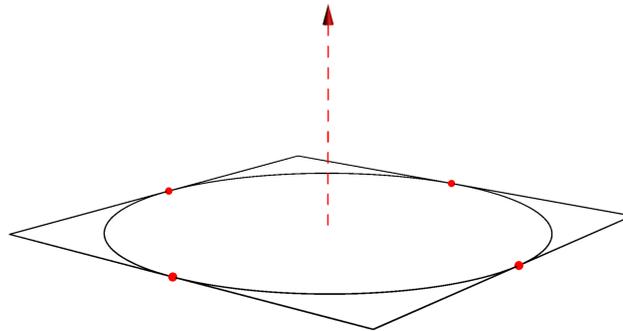
and then distorted into the saddle
`(1,0,0)..(0,1,1)..(-1,0,0)..(0,-1,1)..cycle`:
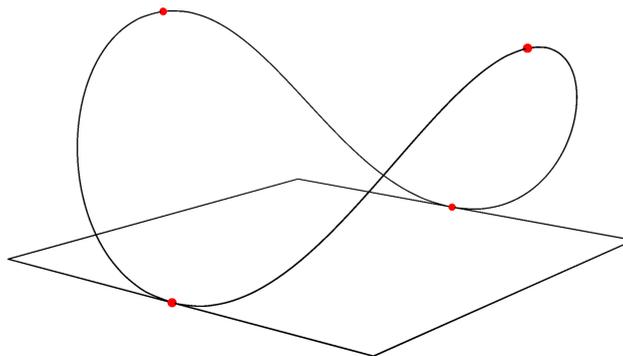
Lifting TeX to 3D

- Glyphs are first split into simply connected regions and then decomposed into planar Bézier surface patches [BS09, SB10]:
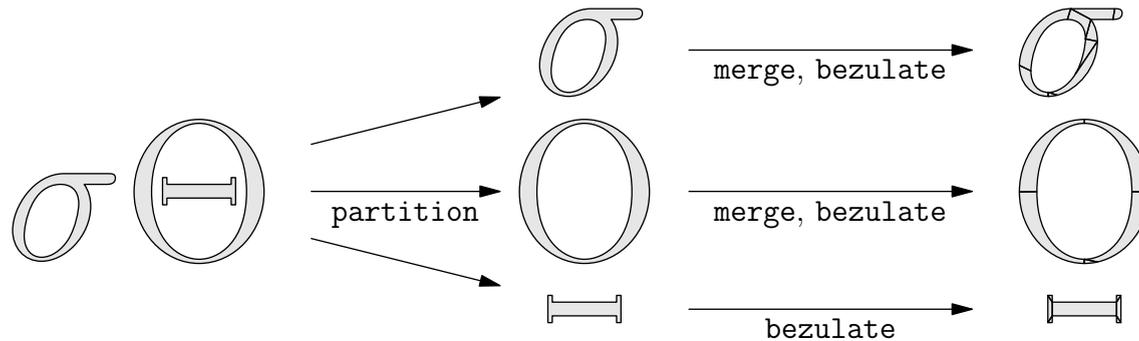
Label Manipulation

- They can then be extruded and/or arbitrarily transformed:

$$\int_{-\infty}^{+\infty} e^{-\alpha x^2}\, dx = \sqrt{\frac{\pi}{\alpha}}$$

Billboard Labels

Embedded

Billboard
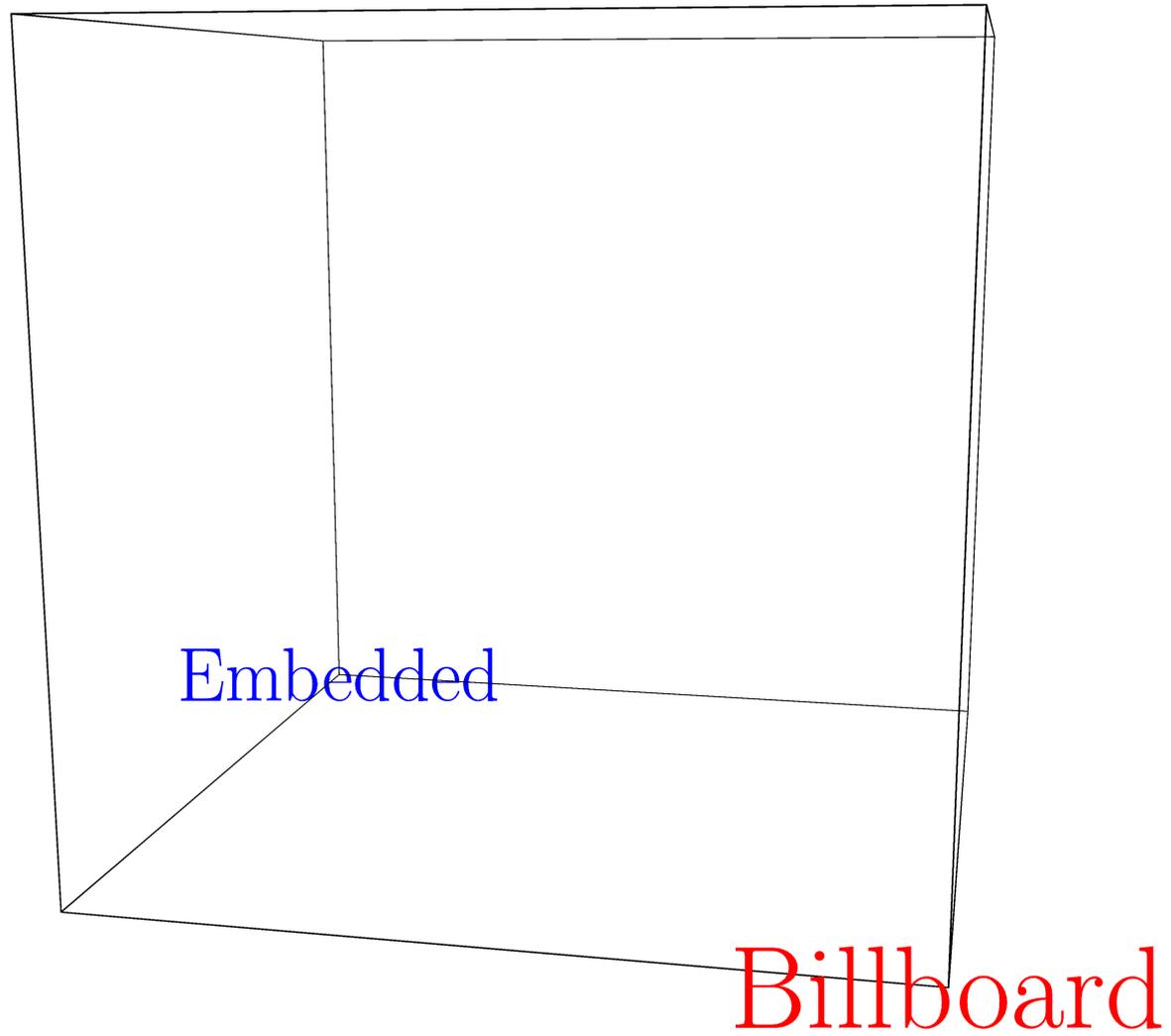
Smooth 3D surfaces
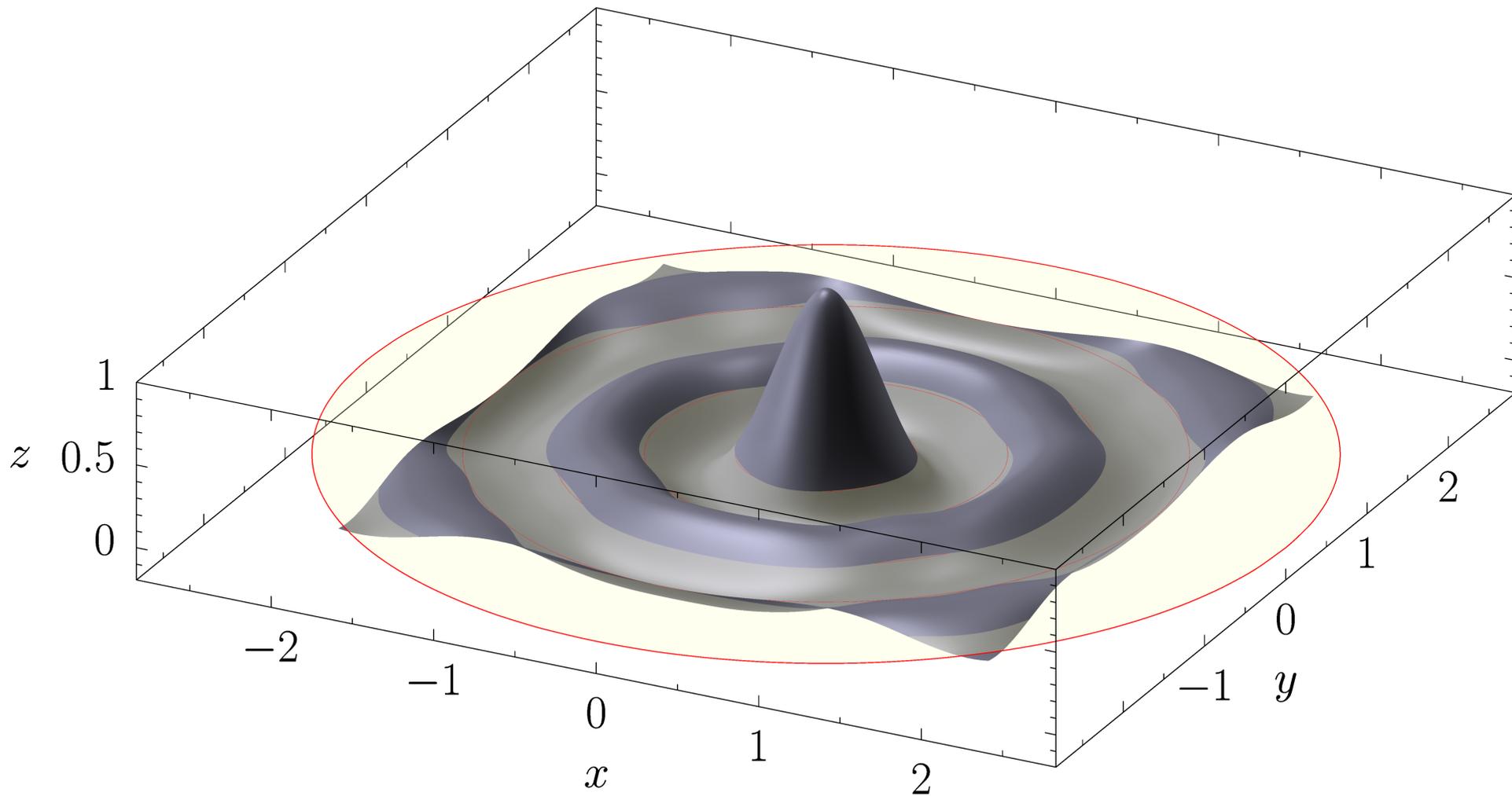
Curved 3D Arrows
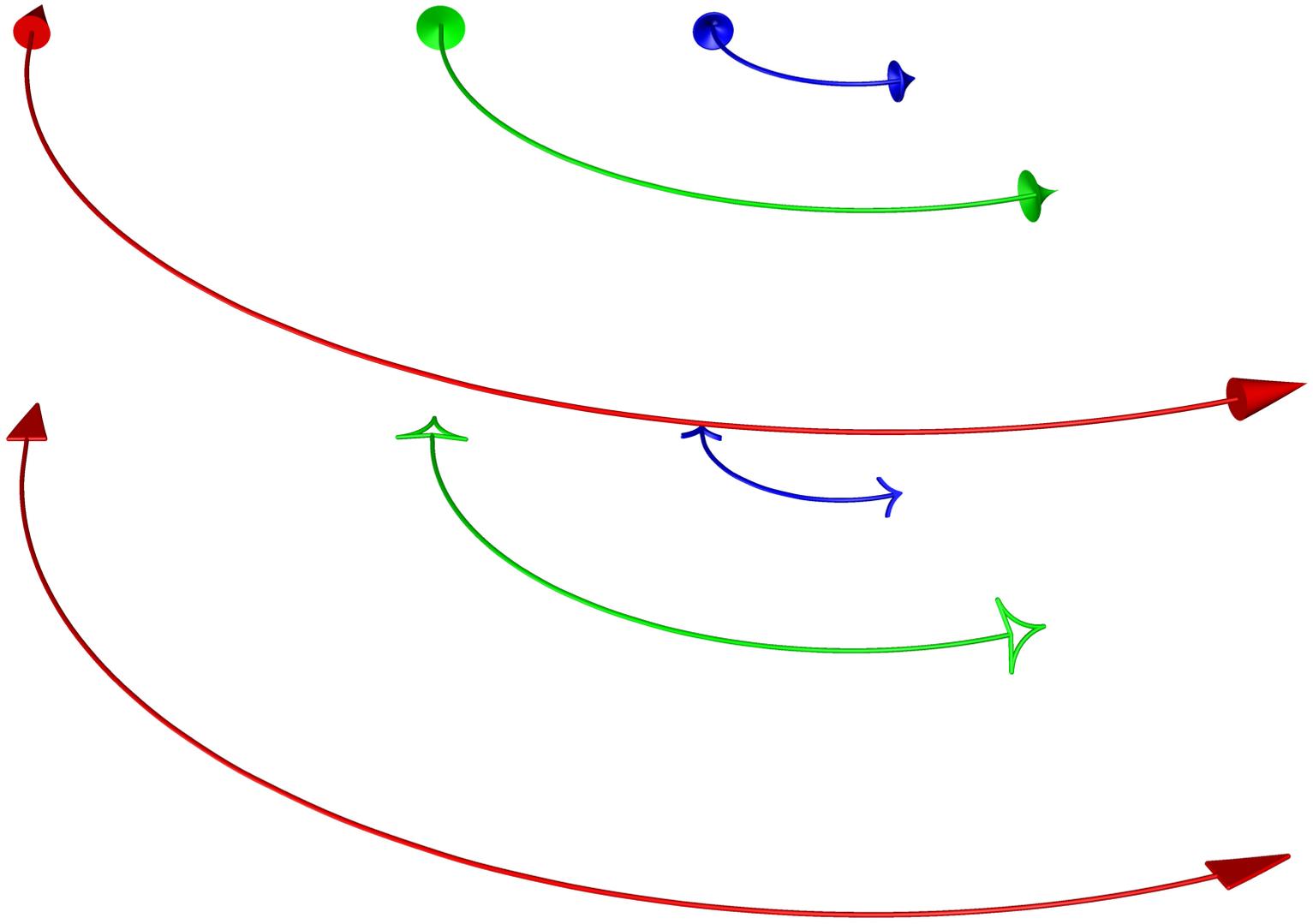
Slide Presentations

- Asymptote has a module for preparing slides.

- It even supports embedded high-resolution PDF movies.

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

Automatic Sizing

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

$(a, 0)$ $(2a, 0)$

`size(0,50);`

$(a, 0)$ $(2a, 0)$

`size(0,100);`

$(a, 0)$ $(2a, 0)$

`size(0,200);`
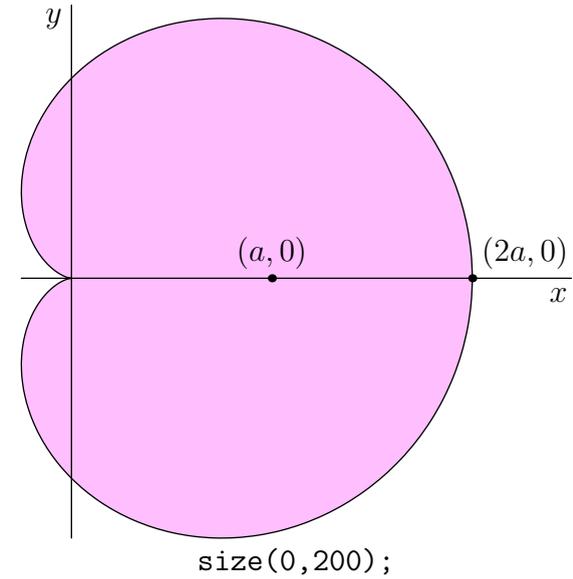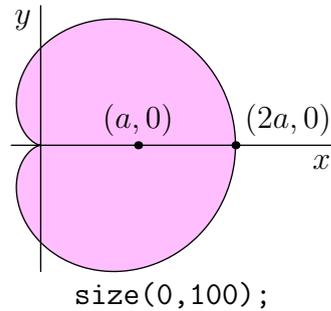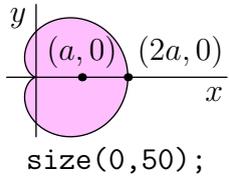
Deferred Drawing

- We can't draw a graphical object until we know the scaling factors for the user coordinates.

- Instead, store a function that, 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 the sum of user and true-size coordinates:
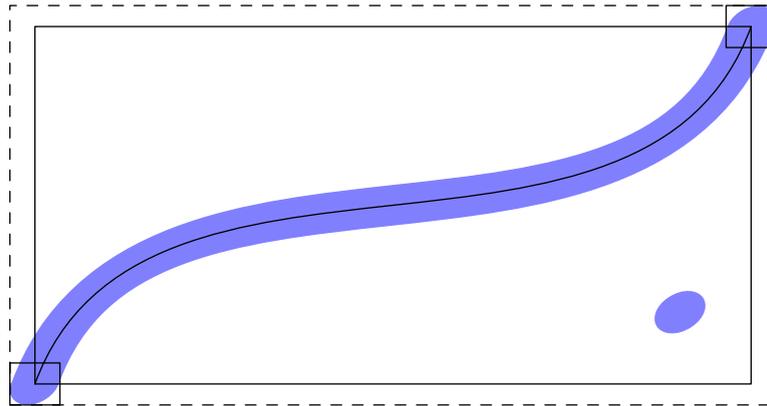

```
pic.addPoint(min(g),min(p));
pic.addPoint(max(g),max(p));
```

- Filling ignores the pen width:

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

- Communicate with LaTeX *via* a pipe 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:

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

- Maximize the variable a subject to a number of inequalities.

- Use the simplex method to solve the resulting linear programming problem.

Infinite Lines

- Deferred drawing allows us to draw infinite lines.

```
drawline(P, Q);
```
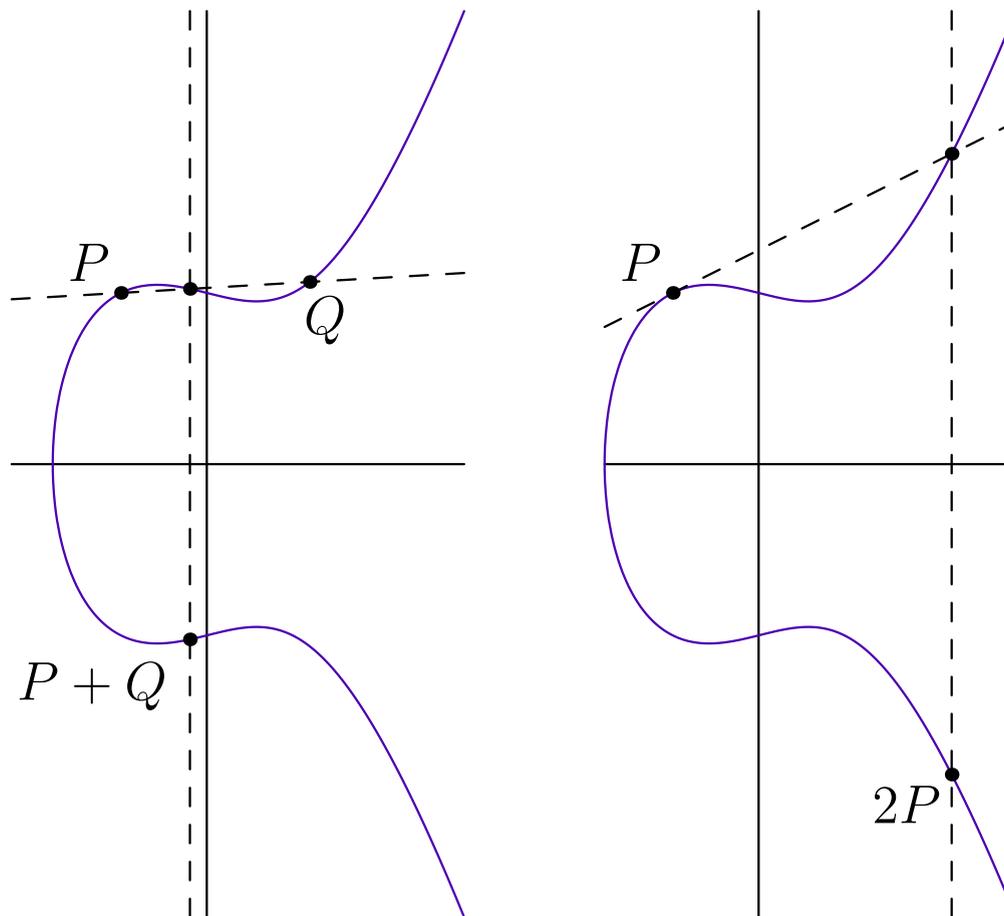
Helpful Math Notation

- Integer division returns a **`real`**. Use **`quotient`** for an integer result:

  ```
  3/4 == 0.75            quotient(3,4) == 0
  ```

- Caret for real and integer exponentiation:

  ```
  2^3     2.7^3     2.7^3.2
  ```

- Many expressions can be implicitly scaled by a numeric constant:

  ```
  2pi     10cm     2x^2     3sin(x)     2(a+b)
  ```

- Pairs are complex numbers:

  ```
  (0,1)*(0,1) == (-1,0)
  ```

Function Calls

- Functions can take default arguments in any position. Arguments are matched to the first possible location:

```
void drawEllipse(real xsize=1, real ysize=xsize, pen p=blue) {
  draw(xscale(xsize)*yscale(ysize)*unitcircle, p);
}

drawEllipse(2);
drawEllipse(red);
```
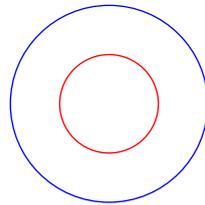
- Arguments can be given by name:

```
drawEllipse(xsize=2, ysize=1);
drawEllipse(ysize=2, xsize=3, green);
```
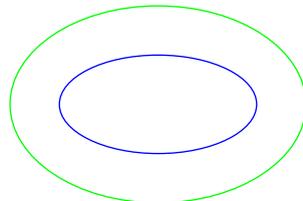
High-Order Functions

- Functions are first-class values. They can be passed to other functions:

```
import graph;
real f(real x) {
    return x*sin(10x);
}
draw(graph(f,-3,3,300),red);
```
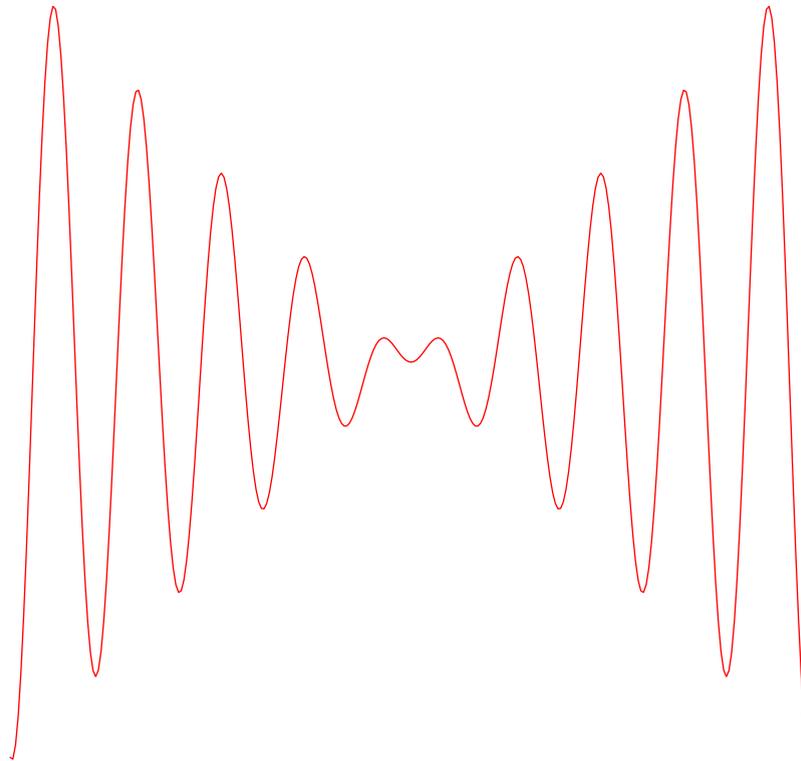
Object-Oriented Programming

- Functions are defined for each instance of a structure.

```
struct Quadratic {
  real a,b,c;
  real discriminant() {
    return b^2-4*a*c;
  }
  real eval(real x) {
    return a*x^2 + b*x + c;
  }
}
```

- This allows us to construct "methods" which are just normal functions declared in the environment of a particular object:

```
Quadratic poly=new Quadratic;
poly.a=-1; poly.b=1; poly.c=2;

real f(real x)=poly.eval;
real y=f(2);
draw(graph(poly.eval, -5, 5));
```

Specialization

- Can create specialized objects just by redefining methods:

```
struct Shape {
    void draw();
    real area();
}

Shape rectangle(real w, real h) {
  Shape s=new Shape;
  s.draw = new void () {
                      fill((0,0)--(w,0)--(w,h)--(0,h)--cycle); };
  s.area = new real () { return w*h; };
  return s;
}

Shape circle(real radius) {
  Shape s=new Shape;
  s.draw = new void () { fill(scale(radius)*unitcircle); };
  s.area = new real () { return pi*radius^2; }
  return s;
}
```

Overloading

- Consider the code:

```
int x1=2;
int x2() {
  return 7;
}
int x3(int y) {
  return 2y;
}

write(x1+x2());  // Writes 9.
write(x3(x1)+x2());  // Writes 11.
```

Overloading

- **x1**, **x2**, and **x3** are never used in the same context, so they can all be renamed **x** without ambiguity:

```
int x=2;
int x() {
  return 7;
}
int x(int y) {
  return 2y;
}

write(x+x());   // Writes 9.
write(x(x)+x());   // Writes 11.
```

- Function definitions are just variable definitions, but variables are distinguished by their signatures to allow overloading.

Operators

- Operators are just syntactic sugar for functions, and can be addressed or defined as functions with the **operator** keyword.

```
int add(int x, int y)=operator +;
write(add(2,3));  // Writes 5.


// Don't try this at home.
int operator +(int x, int y) {
  return add(2x,y);
}
write(2+3);  // Writes 7.
```

- This allows operators to be defined for new types.

Operators

- Operators for constructing paths are also functions:

  `a.. controls b and c .. d--e`
  is equivalent to

  `operator --(operator ..(a, operator controls(b,c), d), e)`
- This allowed us to redefine all of the path operators for 3D paths.

Summary

- Asymptote:

    – uses IEEE floating point numerics;

    – uses C++/Java-like syntax;

    – supports deferred drawing for automatic picture sizing;

    – supports Grayscale, RGB, CMYK, and HSV colour spaces;

    – supports PostScript shading, pattern fills, and function shading;

    – can fill nonsimply connected regions;

    – generalizes MetaPost path construction algorithms to 3D;

    – lifts TeX to 3D;

    – supports 3D billboard labels and PDF grouping.

References

[BH08]   John C. Bowman and Andy Hammerlindl. Asymptote: A vector graphics language. *TUGboat: The Communications of the TEX Users Group*, 29(2):288–294, 2008.

[Bow07] John C. Bowman. The 3D Asymptote generalization of MetaPost Bézier interpolation. *Proceedings in Applied Mathematics and Mechanics*, 7(1):2010021–2010022, 2007.

[BS09]   John C. Bowman and Orest Shardt. Asymptote: Lifting TEX to three dimensions. *TUGboat: The Communications of the TEX Users Group*, 30(1):58–63, 2009.

[Hob86] John D. Hobby. Smooth, easy to compute interpolating splines. *Discrete Comput. Geom.*, 1:123–140, 1986.

[Knu86] Donald E. Knuth. *The METAFONTbook*. Addison-Wesley, Reading, Massachusetts, 1986.

[SB10]   Orest Shardt and John C. Bowman. Surface parametrization of nonsimply connected planar Bézier regions. *Submitted to Computer Aided Design*, 2010.

Asymptote: 2D & 3D Vector Graphics Language

`http://asymptote.sf.net`

(freely available under the LGPL license)