3D Graphics in Sage

Robert Bradshaw

12 August 2008

Brief History

3D Graphics in Sage was a long time in comming

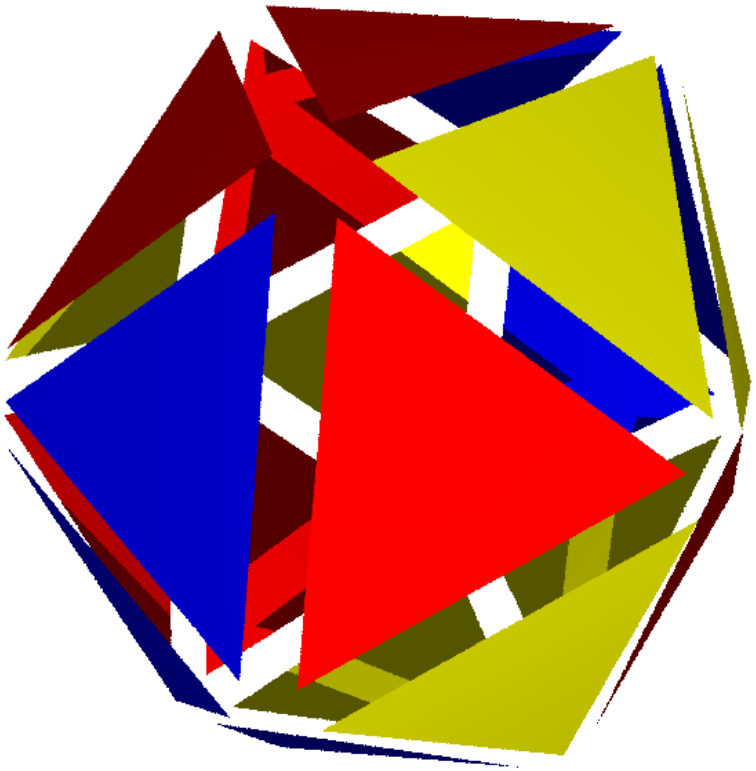
- summer 2006 — tachyon, notebook
- spring 2007 — first attempt (xj3d)
- summer 2007 — second attempt (java3d)
- December 2007 — discover jmol
- August 2008 — Sage Days 9

Status

- Now we have a stable, working core to build on
- Jmol used from command line and notebook
- still lots of unimplemented features

William Stein's talk will be about the front end, I will focus on the underlying implementation.

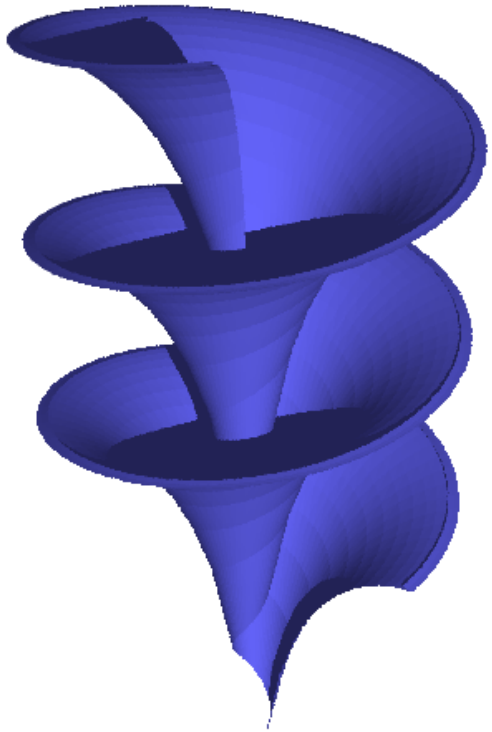
IndexFaceSet



- bread and butter 3D graphics
- represent triangulated graphics
  - suboptimal for jmol, tachyon
- list of vertices
- list of faces by *indexing* into vertex list

ParametricSurface

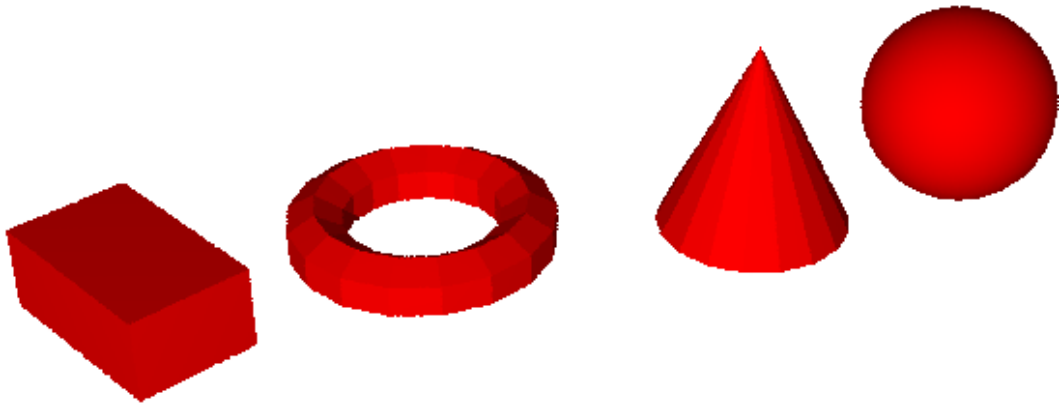
- based on `IndexFaceSet`
- easy to implement
  - override `get_grid()`
  - override `eval()` or `eval_c`
  - or pass into `__init__`
- automatically handles triangulation, bounding box, is_enclosed, etc.



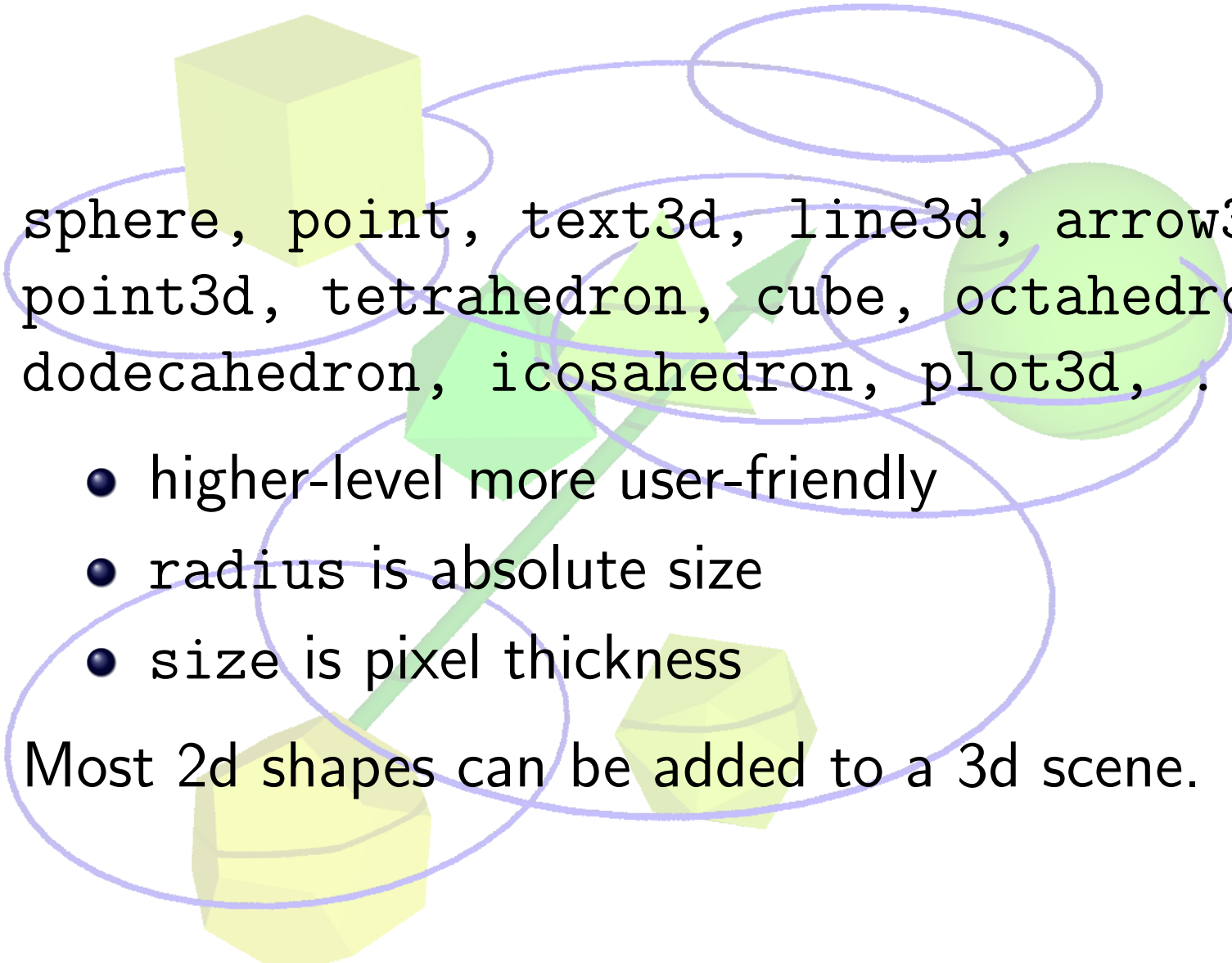
Other Primitives

`Sphere, Cone, Cylinder, Torus, LineSegment, Box`

- based on `ParametricSurface`
- for efficiency, override more methods



Still more shapes



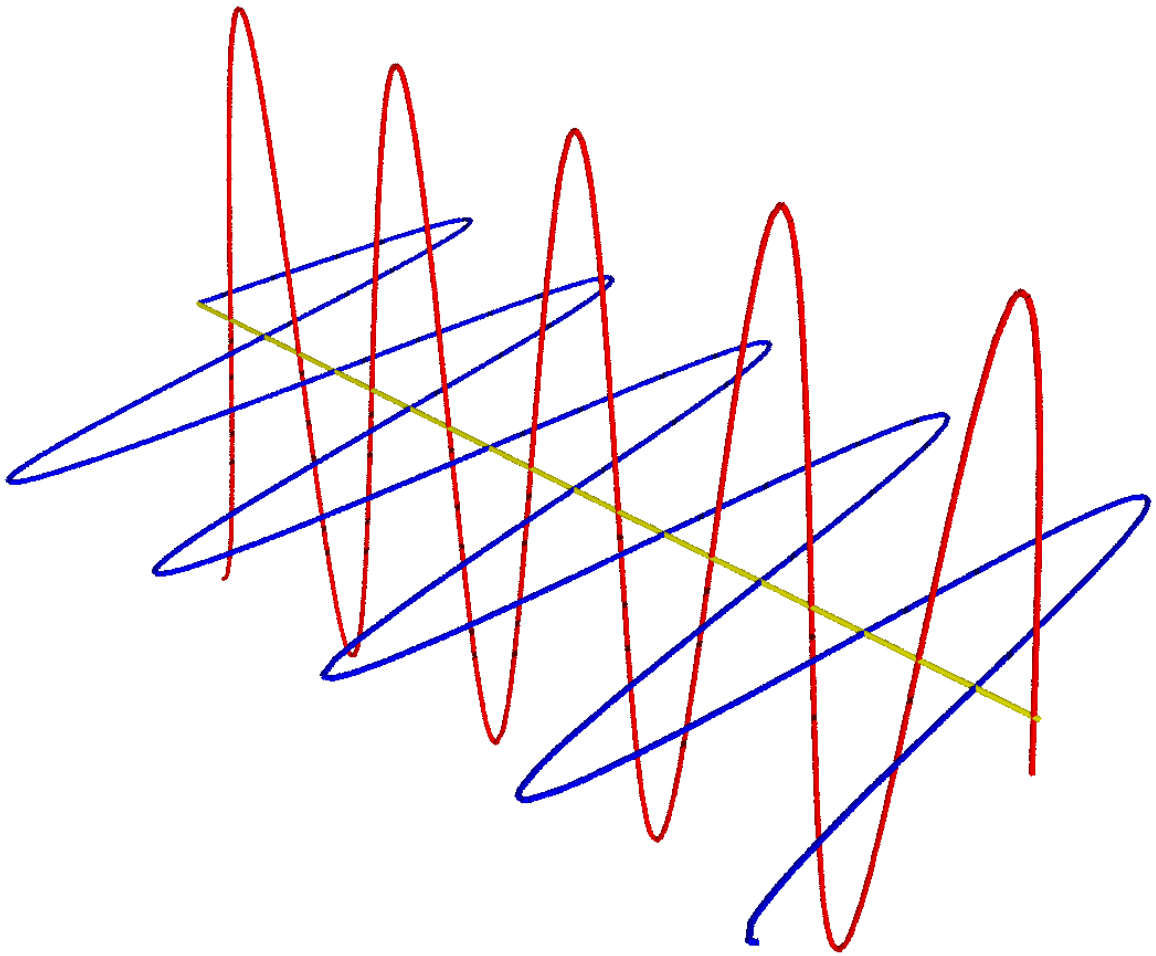
`sphere`, `point`, `text3d`, `line3d`, `arrow3d`, `point3d`, `tetrahedron`, `cube`, `octahedron`, `dodecahedron`, `icosahedron`, `plot3d`, ...

- higher-level more user-friendly
- `radius` is absolute size
- `size` is pixel thickness

Most 2d shapes can be added to a 3d scene.

Still more shapes

```
sage:  g = plot(10*sin(x), (x, 0, 30)).plot3d()
sage:  g += plot(10*cos(x), (x, 0, 30), color='red').plot3d().rotate([1,0,0], pi/2)
sage:  g += line3d([(0,0,0), (30,0,0)], color='yellow')
```



Scenes

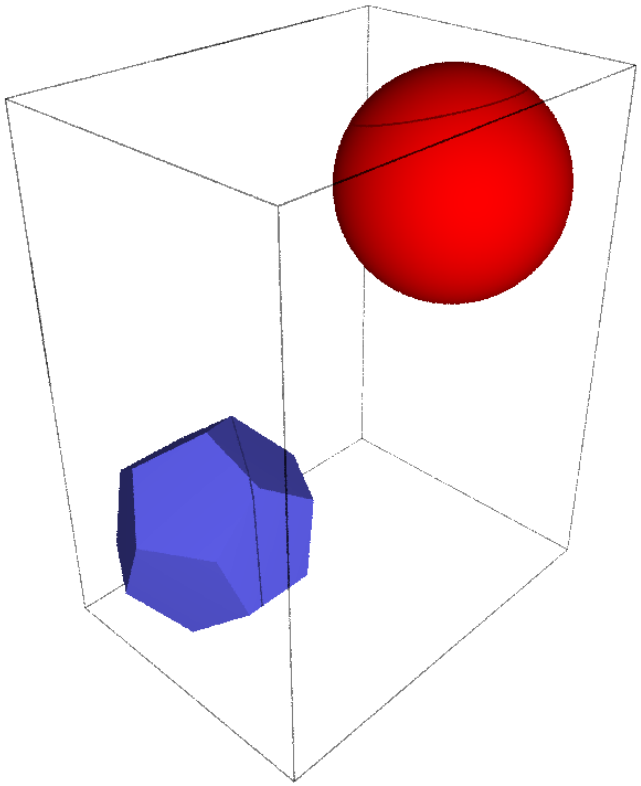
Objects in a 3D scene are laid out in a tree-like structure. There are three kinds of nodes:

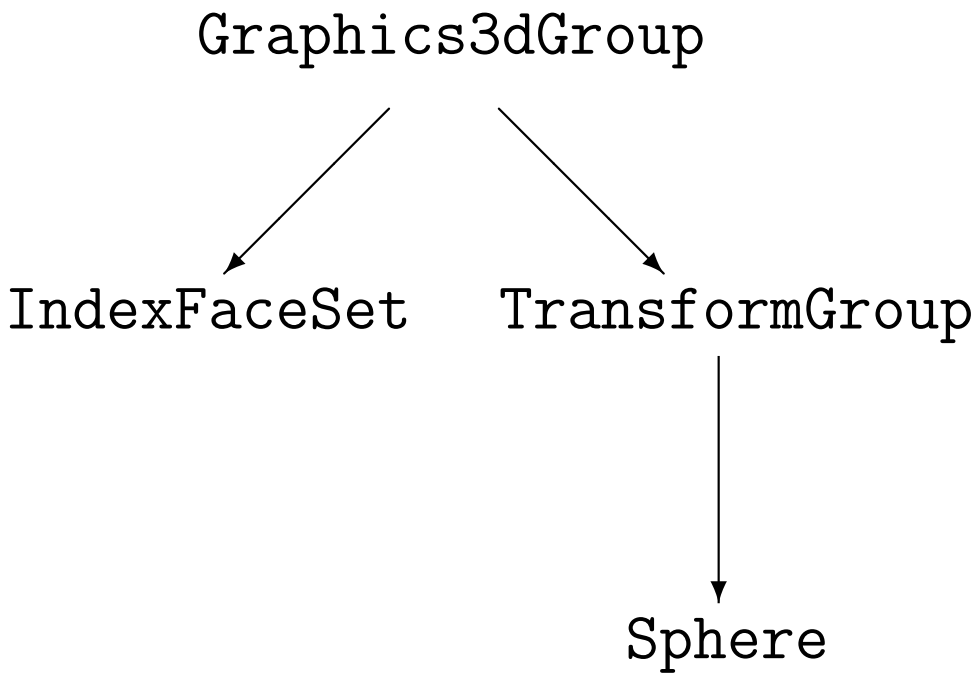
- Graphics3DGroup
- TransformGroup — an affine transform of everything under it
- PrimitiveObject — an actual shape, as exemplified above

TransformGroups are created implicitly on `scale()`, `translate()`, or `rotate()`.

A simple scene

```
sage:  dodecahedron() + sphere((1,2,3), color='red')
```





Rendering

Output to several formats:

- x3d
- jmol
- obj
- Tachyon

Rendering

- rendering is done as a depth-first traversal of the tree
- a single object, `RenderParams` is passed which holds rendering state
- override `x3d_str()` or `x3d_geometry()`, call `x3d()`
- may produce several files

Rendering

Color information is stored in a Texture object

- every object has an associated texture
- texture holds color, opacity, reflective effects, etc.
- texture knows how to render itself

Putting it all together

Currently, the default viewer is Jmol

- all data (including driving script) is packed into a single .zip archive
- applet and app have constant rendering
- applet very customizable with javascript

Pure Java — may not be as fast as hardware accelerated graphics, but it's fast enough and works.

Todo list

There is still *lots* to do

- better X3D support
- textures and bitmaps
- functions of complex variables
- viewpoints and lighting
- interactive feedback
- bugs
- ...