Agile2D: implementing Graphics2D over OpenGL

Jean-Daniel Fekete
INRIA Futurs/LRI
http://www.lri.fr/~fekete

Implemented by
Jon Meyer, Ben Bederson and Jean-Daniel Fekete
for the University of Maryland

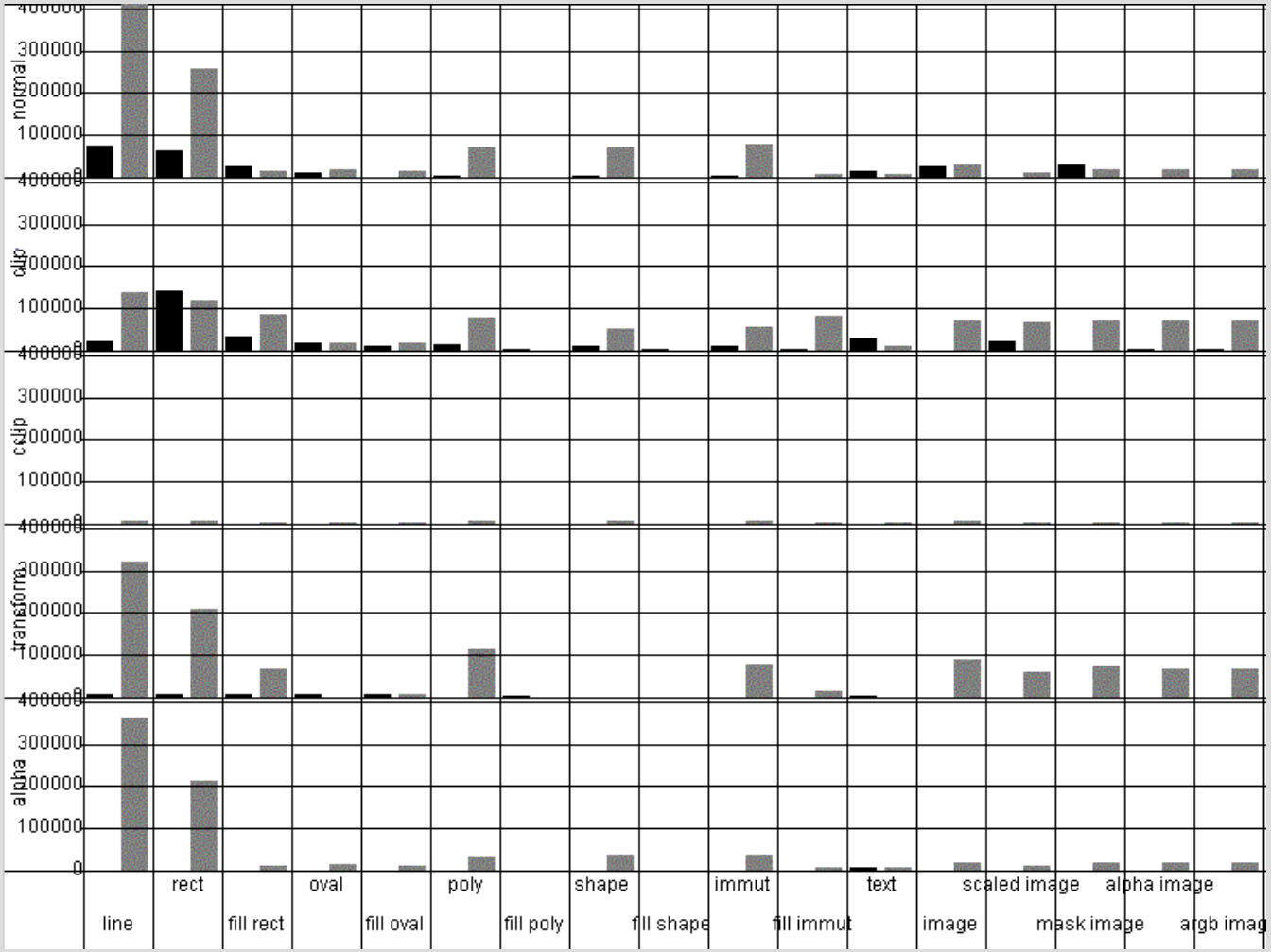
Agile2D Goal

- Provide a Free, Fast and Portable implementation of Java Graphics2D
  - OpenGL is available on more platforms than Java
    - Windows, Linux, MacOS, IRIX, Solaris, BeOS
    - Now on smaller platforms with OpenGL tiny[check]
- Target for Information Visualization applications
- Allow for convenient integration of Java Graphics2D and OpenGL
- New standard Java binding for OpenGL
  - JOGL Free software, BSD licence
  - Time to make it mainstream

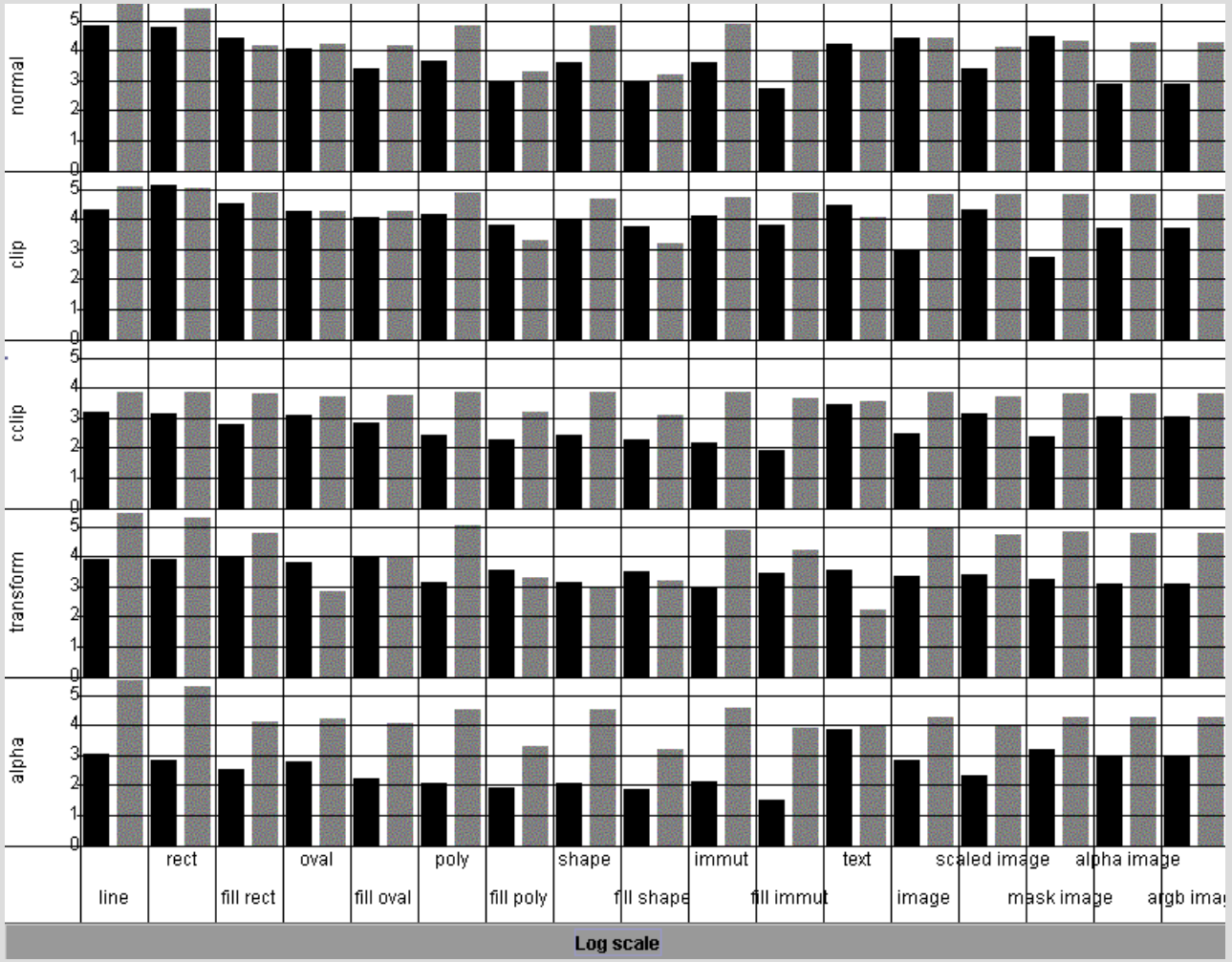
Principles of OpenGL

- Intermediate API
  - not low-level, not high-level
- Expose part of the hardware pipeline
- Provide abstractions to use it at full speed
  - Sometimes 100x faster than software rendering
- Implementations vary from full software (e.g. with Mesa) or full hardware (SGI, 3DLabs, Nvidia, ATI)
- Still evolves, last version is 1.5 on its way to 2.0

Sun vs. Agile2D Performances



Sun vs. Agile2D Performances LOG



Agile2D Implementation

- Relies on Java2D
  - Path, Stroke, Shape, Font Renderer
- Three subsystems
  - Shape rendering
    - Tesselation and rendering attributes
  - Image rendering
    - Texture oriented
  - Font rendering
    - Texture oriented + cache
- Transforms and Clip managed by OpenGL
- Extensions: vertex arrays, OpenGL access
- Missing features: OpenGL based images+AA

AGLGraphics2D

- AGLGraphics delegate all methods to a GraphicsEngine
  - Maintains a “current active AGLGraphics”
- AGLGraphics maintain a local state
  - Paint, background Color, Composite, Stroke, Font, RenderingHints, AffineTransform, clipArea
  - Install it when changing the active AGLGraphics
- Cloning a AGLGraphics is cheap
- Changing context can be expensive

Shape Rendering

- Java shapes are defined by general 2D outlines and an interior rule
- OpenGL can only fill convex polygons or simpler shapes: triangles and quadrilaterals
- Should decompose arbitraty polygons
  - Tesselation, implemented by GLU
  - Expensive, but could be improved
  - Shapes can be cached if desired into a “display list”, stored into the OpenGL memory
- Bottlenecks: Tesselation, transmission
- Solution: cacheing

Shape Tesselation Cacheing

- Most Java shapes are immutable (not all)
- Use a IMMUTABLE_SHAPE_HINT to allow shape cacheing
  - 1st time the shape is rendered, it goes through the tesselation process and is sent to OpenGL in a display list
  - Next times, the display list is used
    - No need to tesseslate, nor to send the results
- A WeakHashMap maintains the Shape to Display List association
  - When the shape is garbage collected, the list is freed

Shape attributes

- Color in OpenGL is RGBA, like Java2D
- Transparency works directly (much faster)
- Gradient and textures are implemented using hardware acceleration (glTexGen)
  - Gradient use 1D textures
  - Textures use 2D textures
  - Blazingly fast! 100x faster than Sun's impl.
- Pixel operators defined in OpenGL not implemented in Agile2D (but could)
- User-defined attributes not supported yet
  - Could be using textures

Image Rendering

- 2 image pipelines in OpenGL
  - Direct image rendering (no transform)
  - Texture mapping
- Use direct rendering when possible
- Little problem with Java RGBA order
- Provides image cacheing just like shapes
- IMMUTABLE_IMAGE_HINT
- Textures are limited in size so tile them (512x512 or 1024x1024) when too big
- All images primitives are implemented

Font Rendering

- No font mechanisms in OpenGL
- 2 mechanisms
  - Texture mapping for smaller sized
  - Shapes for larger size
- Currently only Latin1 characters
- Small size = all glyphs fit in a 512x512 texture
- Cache of textures
- Rendering done by Sun's Font Rendered

Missing features

- No OpenGL rendering in images
  - 3 possible methods
    - Offline GL buffers: slow
    - Rendering in back buffer and copy: painful
    - Pbuffers: not portable but fast, not implemented by gl2java but in JOGL
  - Would complete the implementation
  - VolatileImage could be done in one large Pbuffer
- Font management for non latin1 glyphs
  - Use Unicode “blocks” (roughly language)
  - Smaller blocks can use the same mechanism
  - Large blocks should allocate glyphs dynamically
- Anti Aliasing: easy using multi-sampling
- User-defined Paint attributes ???

Extensions

- Vertex Arrays
  - Send simple geometry and attributes directly to OpenGL
  - Very fast: 15/30/45 million triangles per second, with color, texture indices, etc.
- Direct OpenGL rendering
  - Method runGL(GLEventListener ev)
  - Saved the context and restores in afterwards
  - Heavyweight but preserves the OpenGL state for Agile2D

Agenda

- Find funding to port Agile2D to a full port using JOGL
  - 1 person, 6 months
- Port JOGL to SWT and classpath
- Hard problems:
  - Font rendering
  - Implementation of Area (maybe using GLU tesselation)