UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
UNIVERSITY OF SZEGED
Department of Software Engineering

GPU Based 2D Rendering With TyGL

Zoltan Herczeg

12/09/13

Overview

- Tile Based Embedded GPUs
- Key Features of the TyGL Engine
  - Batching Pipeline
  - Trapezoid Based Path Rendering
  - Image Drawing
- Results and Future Work

Tile Based
Embedded GPUs

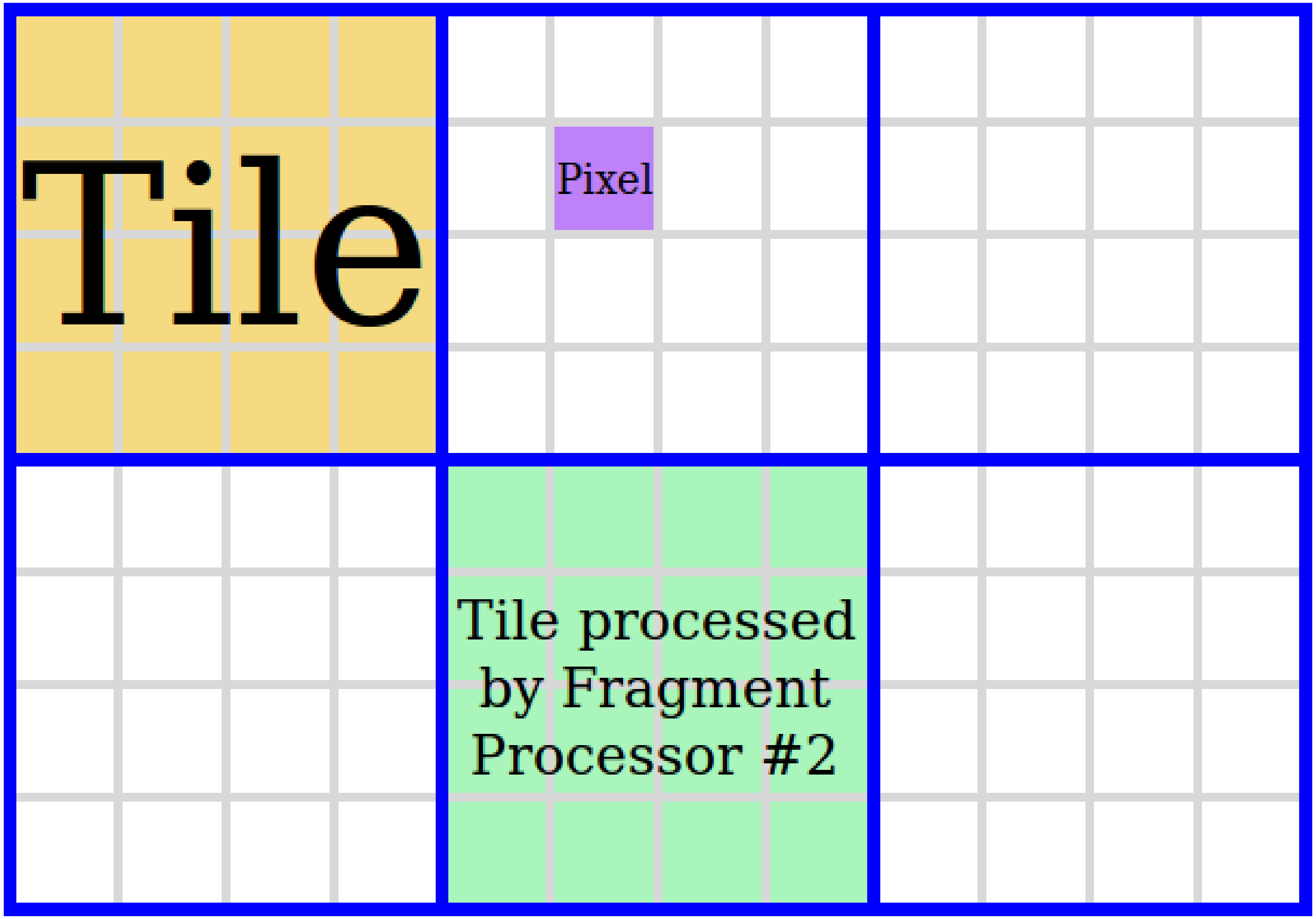
TyGL 2D Rendering Engine

- GPU based 2D rendering engine
  - Without software fallback
- Requires OpenGL-ES 2.0
- Part of WebKit
  - Not an independent library yet (can be)
- Open source
  - https://github.com/szeged/TyGL
- Optimized for embedded GPUs

Tile Based Embedded GPUs

- Unique approach for 3D rendering
- All drawings performed on an internal buffer
  - Usual tile size is 16x16 pixel
  - Each tile is fully drawn by a single fragment processor
  - Contains all pixel data: RGBA color, depth data, stencil data
  - SRAM: very fast reading / writing

Tile Based Rendering



Optimized for 3D Rendering

- Rendering independent frames
  - Pixel data is never reused
  - Full screen redraw (SwapBuffers)
  - Usually composed of several shapes
- The GPU can render the current frame while the CPU sends all draw commands for the next frame

What About 2D Rendering?

- Usually only the changes (dirty regions) are redrawn
  - Most changes are simple
- GPU rendering is very inefficient when doing little work
  - Initializing / flushing GPU pipeline is costly
  - Forcing the GPU to draw a single pixel may result transferring megabytes of tile data (memory transfer is costly)

Challenges of Our TyGL Project

- Reducing Frame Buffer (FBO) switches
  - Changing the current rendering target usually forces the GPU to do work
  - All draw operations should be finished before changing the FBO
- Texture allocation is costly
  - Textures should be reused
  - However, unexpected texture ghosting may have a large overhead

Challenges of Our Project (2)

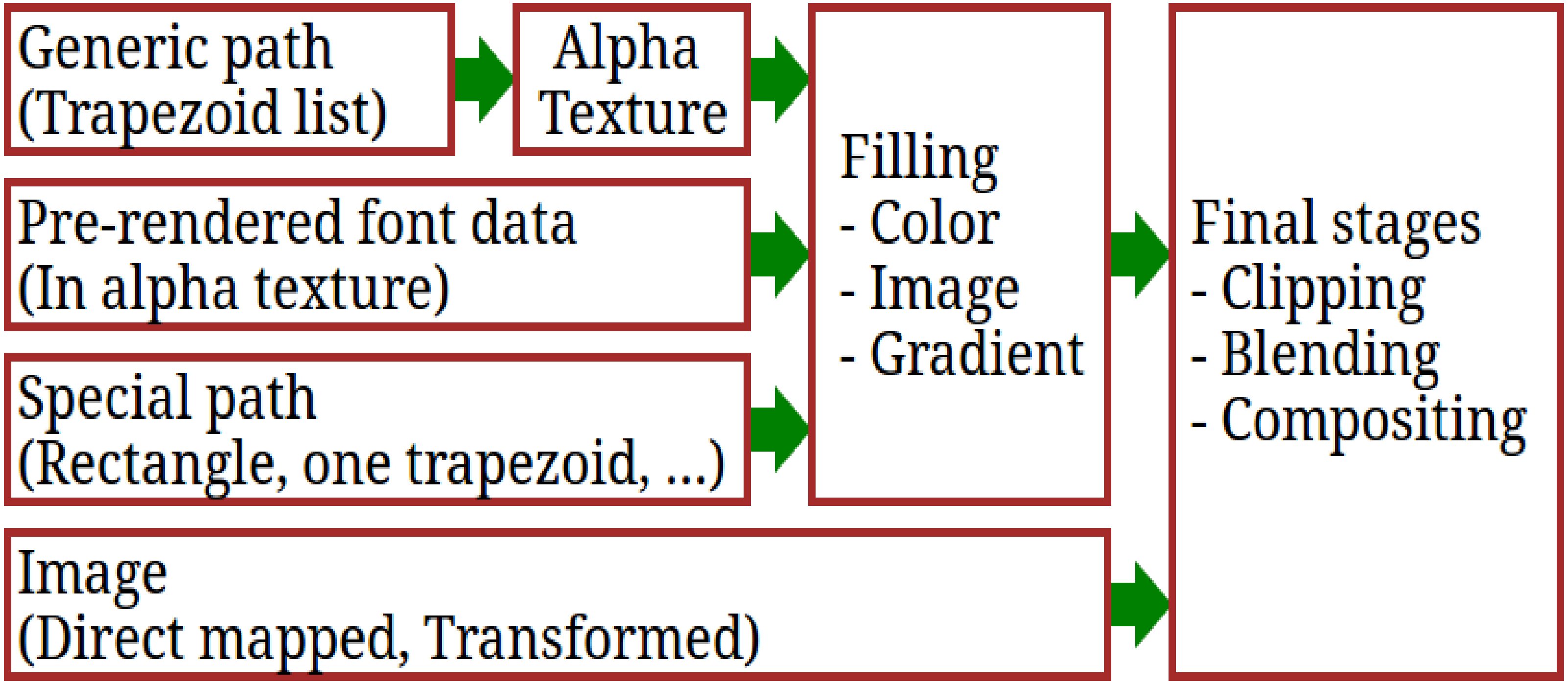
- We have limited control over tiling
  - Scissor boxes are not efficient
  - QCOM_tiled_rendering (Qualcomm specific extension)
- Reduce the number of OpenGL-ES calls
  - OpenGL has a considerable API overhead
  - Batching Pipeline
- Reduce pixel overdraw
  - No work has been done yet

Challenges of Our Project (3)

- Supproting multi process model:
- Inter-process texture sharing
  - X11 Pixmaps, GraphicBuffer on Android
  - Copy data between CPU/GPU is very slow
- Limited high-level synchronization
  - Cross process fencing is not possible, some drivers don't even support cross thread fencing

Key Features of the TyGL engine

2D Batching Pipeline



Generic path
(Trapezoid list)

Alpha
Texture

Pre-rendered font data
(In alpha texture)

Special path
(Rectangle, one trapezoid, ...)

Image
(Direct mapped, Transformed)

Filling
- Color
- Image
- Gradient

Final stages
- Clipping
- Blending
- Compositing

2D Batching Pipeline (2)

- 2D engines has a long (but fixed) pipeline with several stages
- Pixels can be processed in parallel
  - No dependencies between them
  - Can be efficiently implemented on the GPU
- Every stage is applied only once
  - No branches or loops

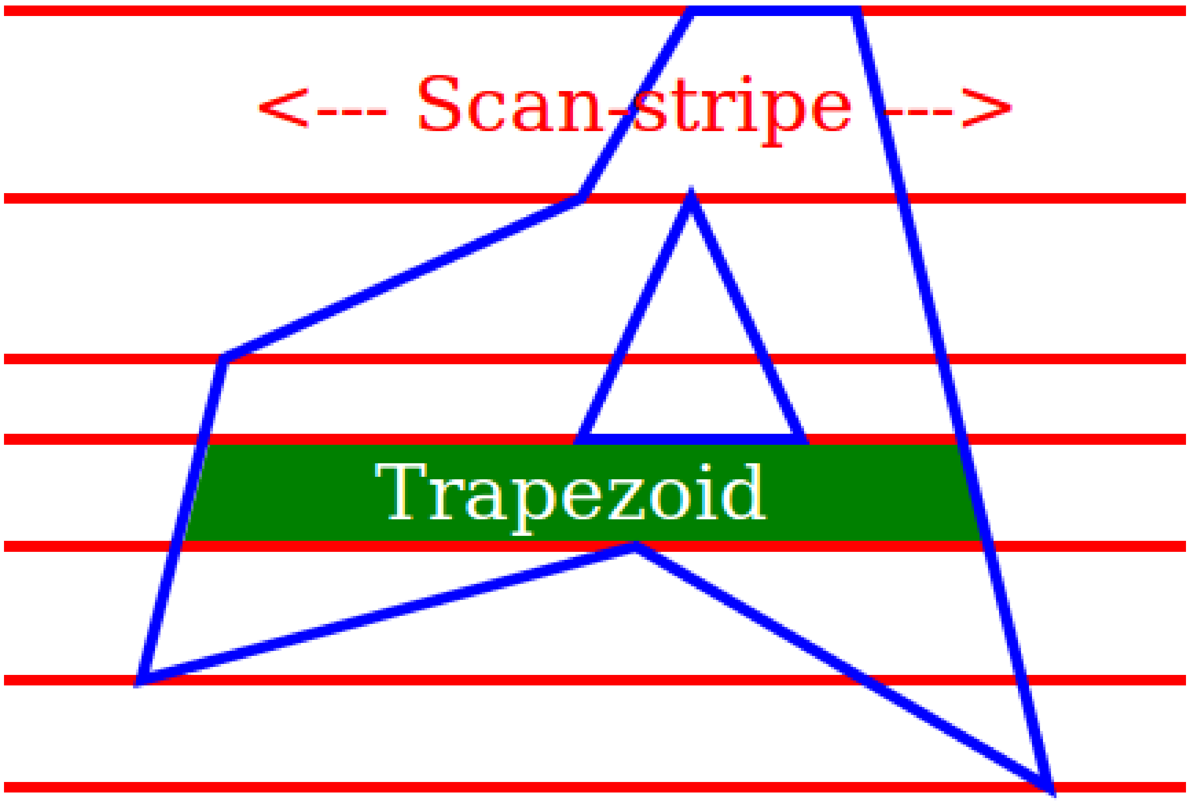
2D Batching Pipeline (3)

- When multiple shapes are drawn with the same shader program, we can render them with a single OpenGL command
  - Called batching
- The shader program represents a 2D pipeline configuration in TyGL

2D Batching Pipeline (4)

- Theoretically we could have a single shader for the whole 2D pipeline, but the code size would be too big with too many configuration arguments
  - GPU shader cache is limited
- We try to balance between batching and pipeline size

Trapezoid Based Path Filling



Comparing FireFox and TyGL

EFL - Cairo

EFL - TyGL

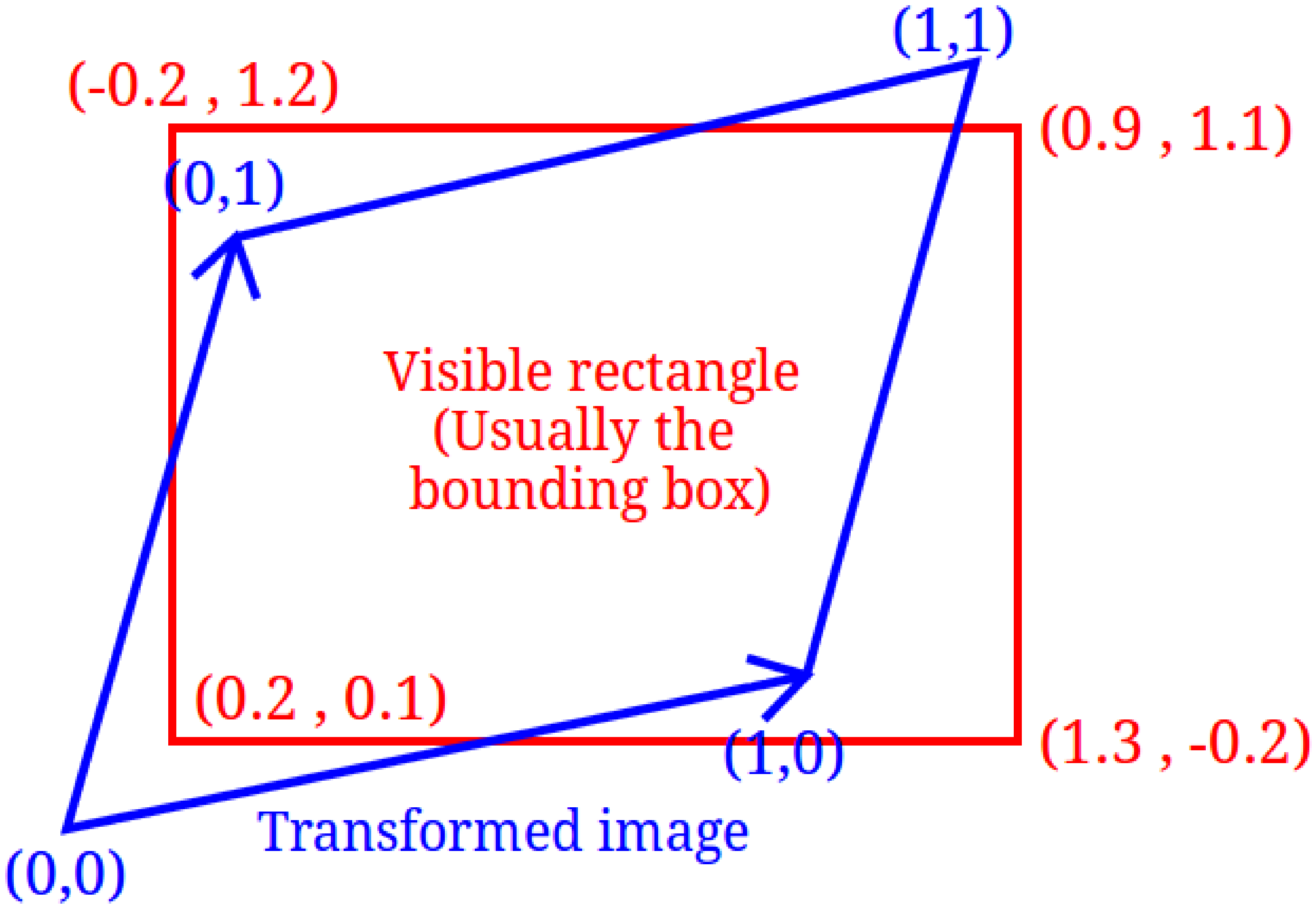
Trapezoid Based Path Filling

- Extending scan-lines to scan-stripes
  - Their height is arbitrary
  - Can contain non-intersecting lines with any slope
- High-quality anti aliasing
- Clipping rectangle can be supported without scissor

Image Drawing

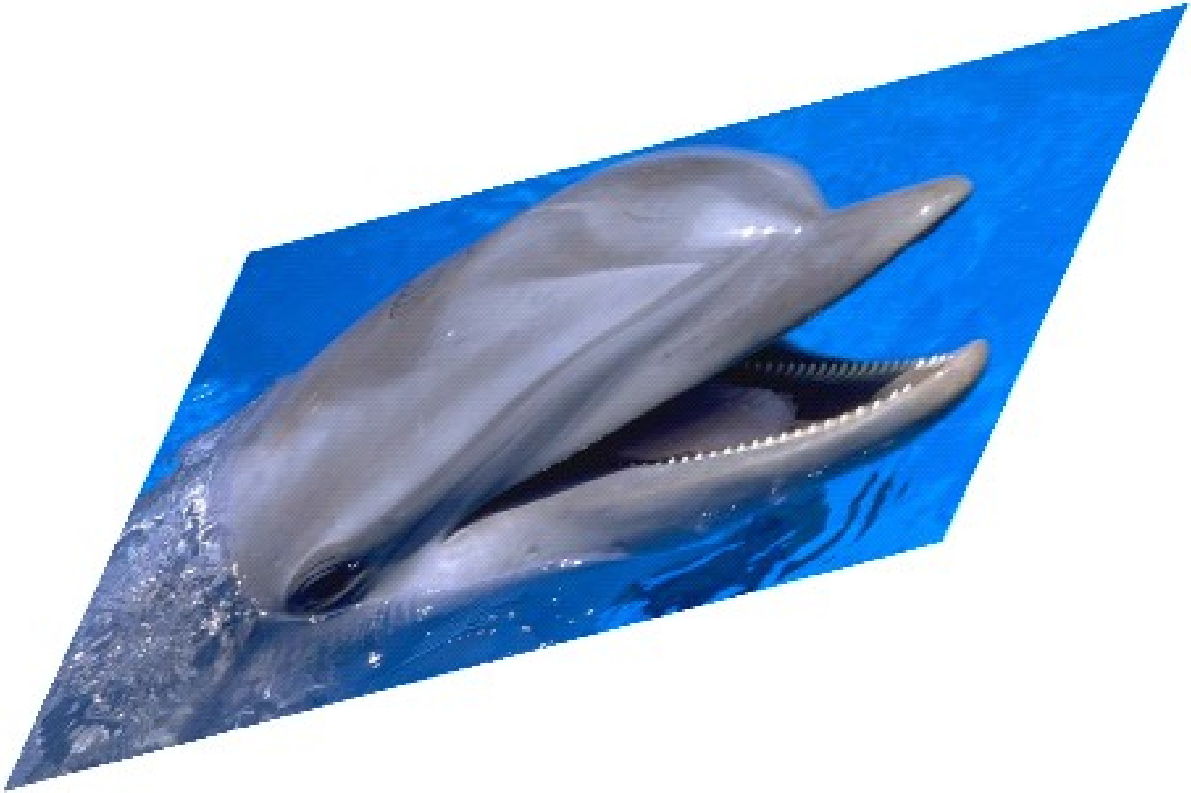
- Drawing anti-aliased images
- Theorem: affine transformation of a parallelogram is always parallelogram
- Barycentric coordinate system
  - Relative coordinates are the linear interpolation of the 4 corner point
    - GPU do this automatically
  - Simplified formula (no dot product)

Image Drawing (2)



UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
UNIVERSITY OF SZEGED
Department of Software Engineering

Image Drawing Example



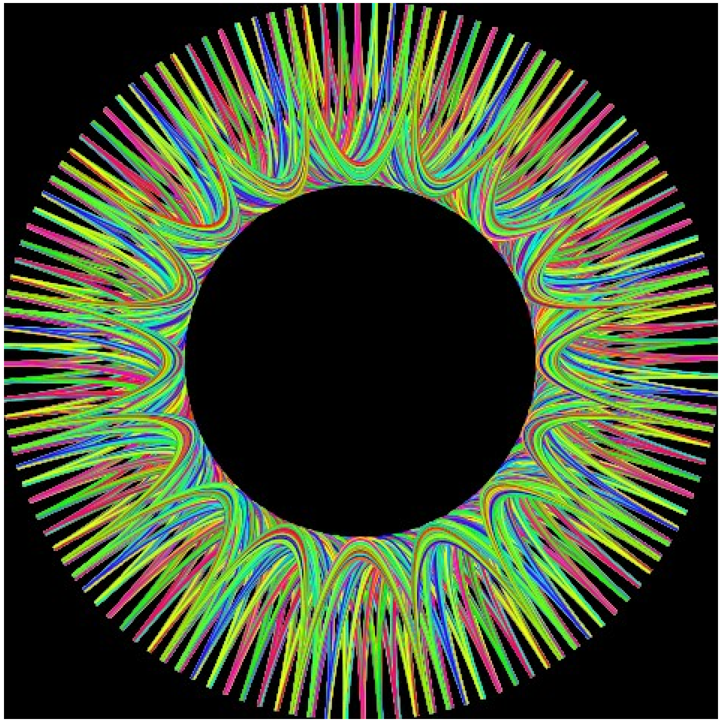
Texture Sharing

- WebKit-EFL uses Coordinated Graphics to transmit updates between processes
- Multiple updates are drawn onto the same texture to reduce memory allocations
  - This is useful for the TyGL engine, since we can draw multiple updates without changing the render target

Results and Future Work

Results (Cairo vs TyGL)

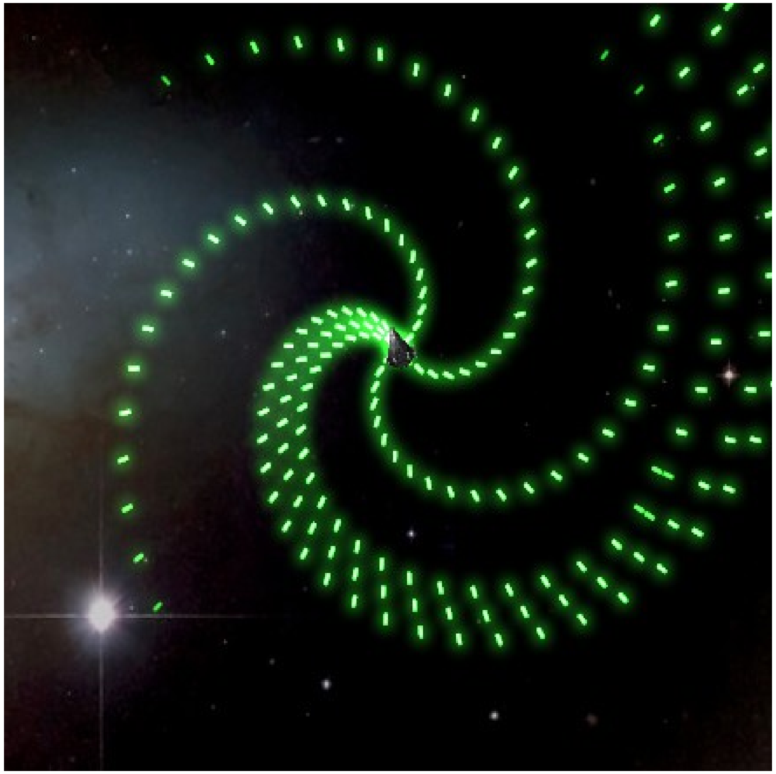
- Compare Cairo-EFL and TyGL-EFL ports
- Canvas-performance test: 1.7 times faster with TyGL
  - (2 tests were skipped)



http://flashcanvas.net/examples/dl.dropbox.com/u/1865210/mindcat/canvas_perf.html

Results (Cairo vs TyGL) (2)

- Asteroids-benchmark: 3.4 times faster with TyGL
  - (2 tests were skipped)



http://www.kevs3d.co.uk/dev/asteroidsbench/

Future: Pixel Local Storage

- Allows direct access to the internal pixel data during tile rendering
  - Multi-vendor support
- This data is preserved when the tile is processed
- Fragment shaders can share data
  - Very useful for path rendering, clipping, blending
- Blocking for 250 ms randomly ???

Summary

- Current GPUs are powerful hardware components, but we are limited by driver side limitations
  - The API is designed for single process classic 3D rendering
- The project is far from reaching its full potential. We need help from driver side
  - Fine-grained control over the device

Thanks for listening