Realtime Graphics

Guest lecture in “Datorgrafik med Interaktion”

by Mikael Kalms

Me, myself and I

Mikael Kalms

33 years old

Studied M.Sc at LiTH

Programmer at EA DICE

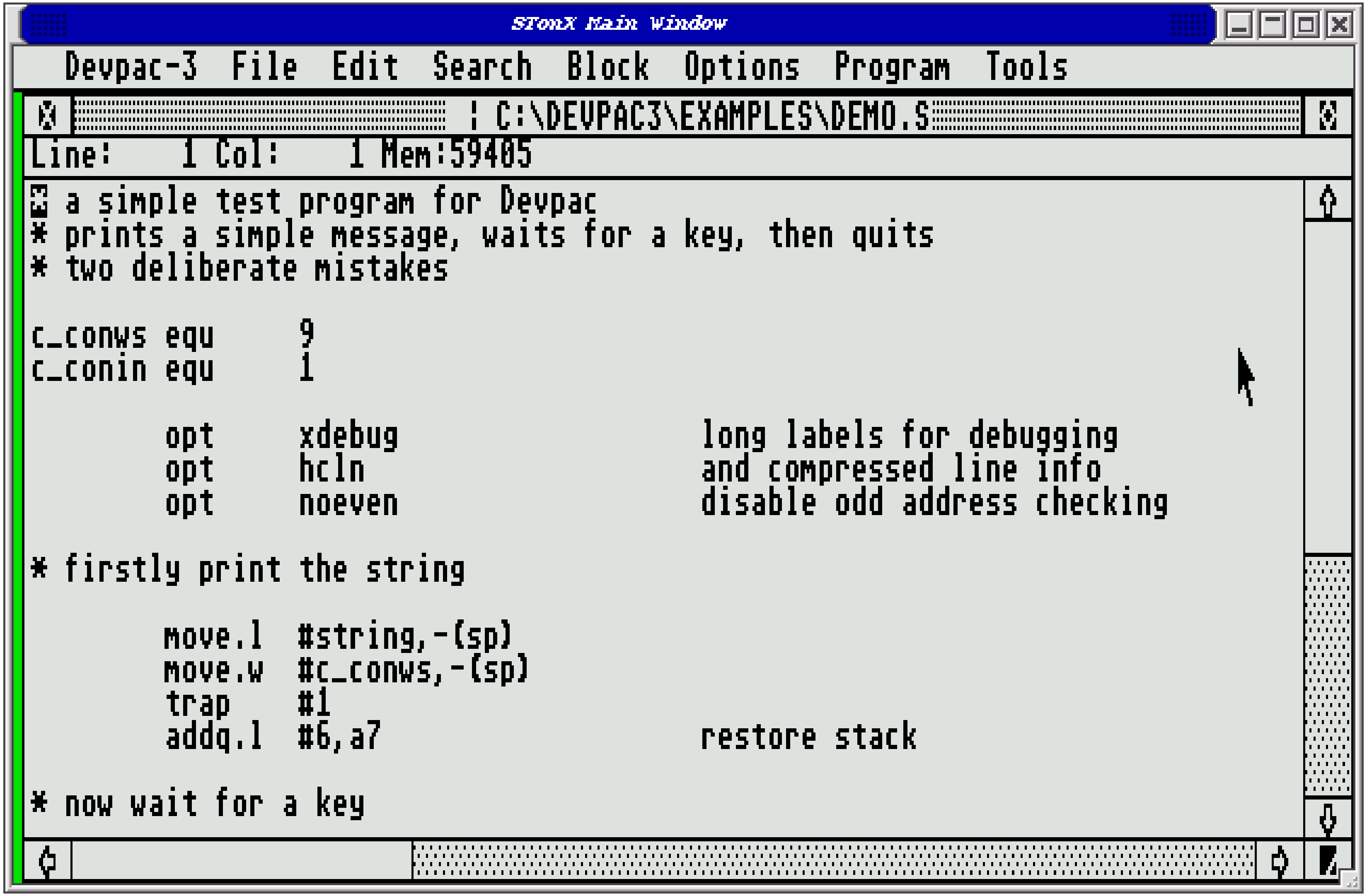
Used to do rendering / systems work

Nowadays more online stuff

Graphics programming is still a hobby of mine!

My formative years

My first assembler



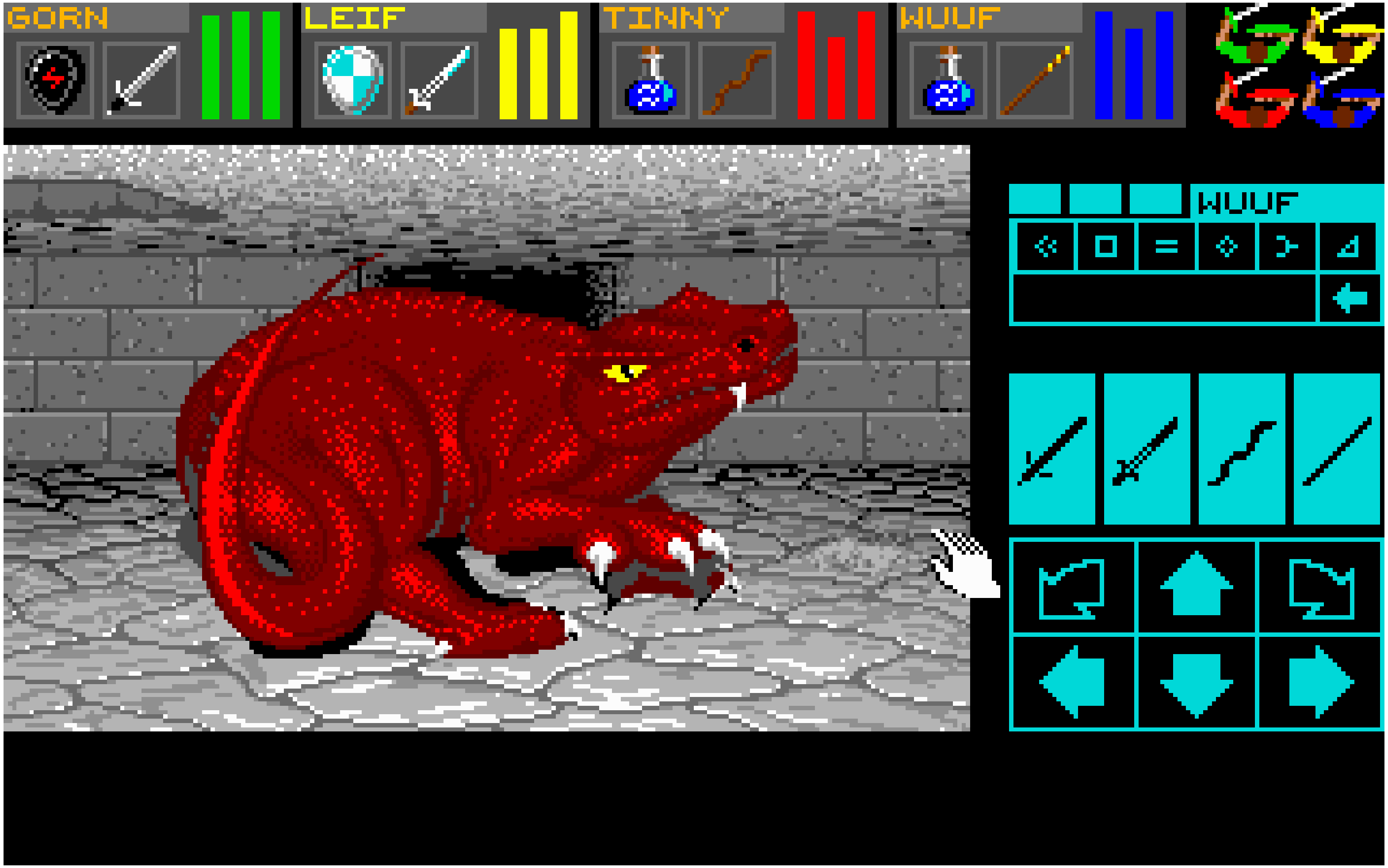
STonX Main Window
Devpac-3 File Edit Search Block Options Program Tools
C:\DEVPAC3\EXAMPLES\DEMO.S
Line: 1 Col: 1 Mem:59405

```
* a simple test program for Devpac
* prints a simple message, waits for a key, then quits
* two deliberate mistakes

c_conws equ     9
c_conin equ     1

        opt     xdebug                  long labels for debugging
        opt     hcln                    and compressed line info
        opt     noeven                  disable odd address checking

* firstly print the string

        move.l  #string,-(sp)
        move.w  #c_conws,-(sp)
        trap    #1
        addq.l  #6,a7                   restore stack

* now wait for a key
```

Back then, this was cool:

http://www.youtube.com/v/BLYCwFjzaaA

Back then, this was fresh:



(Still is, if you ask me...)

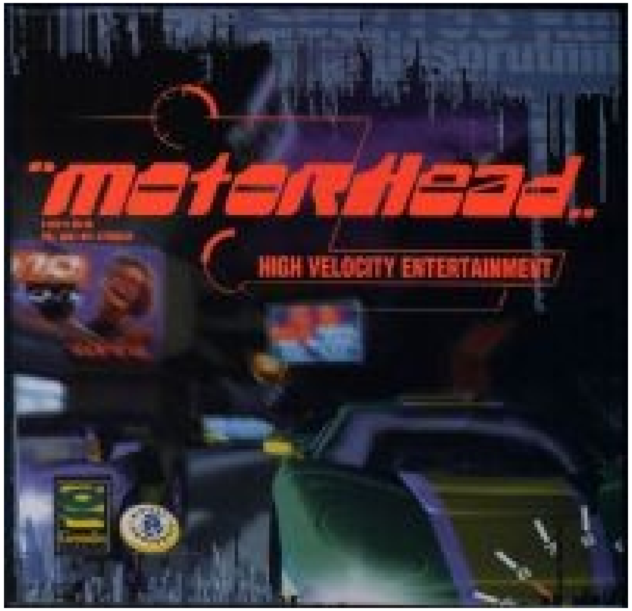
My machine of choice since 1994

+

50MHz is plenty.

http://www.youtube.com/v/4vW0U5zB4JA

I've worked on games.



I've worked on games.

I've worked on games.

I've worked on games.

I've worked on games.

But, back on topic.

Characteristics of realtime graphics

- 10-100 images/second generated (so: high throughput)
- If it's interactive, then latency is also important
- Minimum performance is important; it is often better to have consistent performance, than high peak performance

My approach to realtime graphics programming

50% play around

50% directed work

My choice of language: C++

Why?

- “the most advanced assembler on the market”
- Available on nearly all machines
- Can access native APIs on nearly all OSes
- Control over memory management
- Control over memory layout
- Control over CPU usage
- Control over code generation
- No garbage collection

You can use other languages too, but you are then limiting yourself.

Things to come

Before the break, we will discuss...

Our test harness.

Image processing.

Image bending.

A. Our test harness

- Blank canvas
- What's an Image?
- What's a FloatRGBColor?
- How do RGB colors work?
- How do I draw a pixel?

B. Image processing

- Displaying an image.
- Computations on color values.
- Computations on color components.
- FIR low pass filter.
- IIR low pass filter.

FIR filter: concept

Output pixel = weighted combination of some neighbouring input pixels

IIR filter: concept

Output pixel = weighted combination of some neighbouring input pixels AND some neighbouring output pixels

C. Image bending

- How do we scroll an image?
- Per-line distortion.
- Image zooming (map source → dest)
- Image zooming (map dest → source)

Relax.

It's break time.

Regroup in 15.

Things to come

Rendering filled shapes...

… using per-pixel test

… using scanconversion

… using spantables

And then, filling the shapes with interesting bits

Finally, let's do something interesting with our tools.

E. Per-pixel test

- How do we draw a circle?
- Drawing a halfspace.
- Drawing a triangle.

Circle equation

x^2 + y^2 <= r^2

Halfspace equation

$$Ax + By + C >= 0$$

Triangle equation set

Set up one halfspace eqn per edge.

If halfspace1 && halfspace2 && halfspace3 holds, then (x,y) is inside the triangle.

F. Scanconversion / spantables

- Why scanconversion?
- Draw circle using scanconversion.
- Draw triangle using scanconversion.
- Draw triangle using spantables.

G. Shading our triangles

- Gouraud shaded triangle
- Texturemapped triangle

H. Using our texturemapper

- Zoomrotator.
- Image distorter.
- Tunnel.

The end.

Questions?

Reach me at: mikael@kalms.org