Hogeschool van Amsterdam

3D Game Engines (introduction)

Advanced Graphics Programming

CREATING TOMORROW

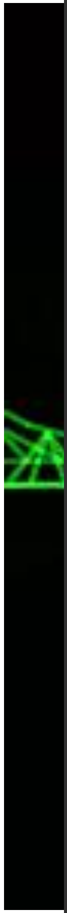
This lesson

- General overview

- Graphics engine within a game (engine)
- A first look at the 3d ‘anatomy’

3d engines / graphics

Short history



Graphics are (still) “big” in industry

- “**~80%** of budget is used for graphics”
  - Money wise
    - “visuals sell”
  - *Processing power* wise
    - Both GPU and CPU

…*lots* of research, time & work into ‘Realtime graphics’ over the years…

- Definitely true in AAA-industry
  - Less in Indie scene… (but still considerable)

The main problem with 3d…

“Creating nice visuals” is easy…
“Creating nice visuals *with good performance*” is hard!!!

Arkham Knight (june 2015)

Good optimization

…is *not* about “just having a good engine”

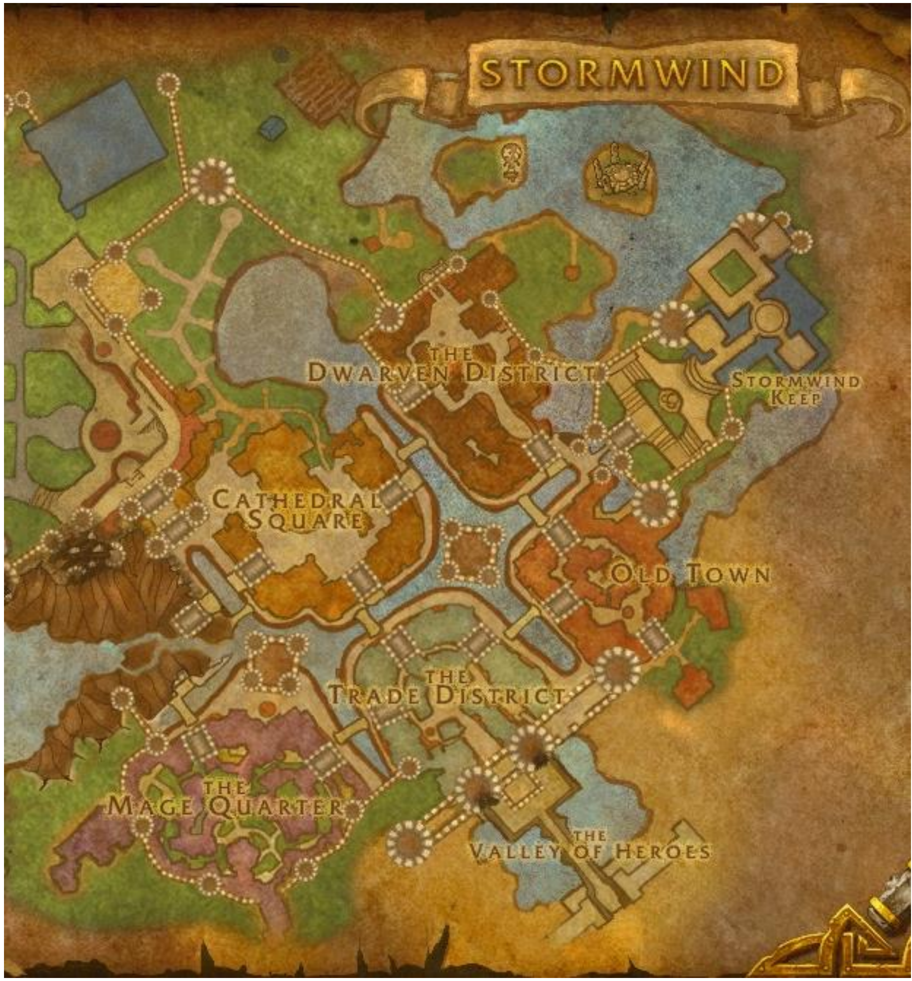
It’s mostly about *specific solutions*:

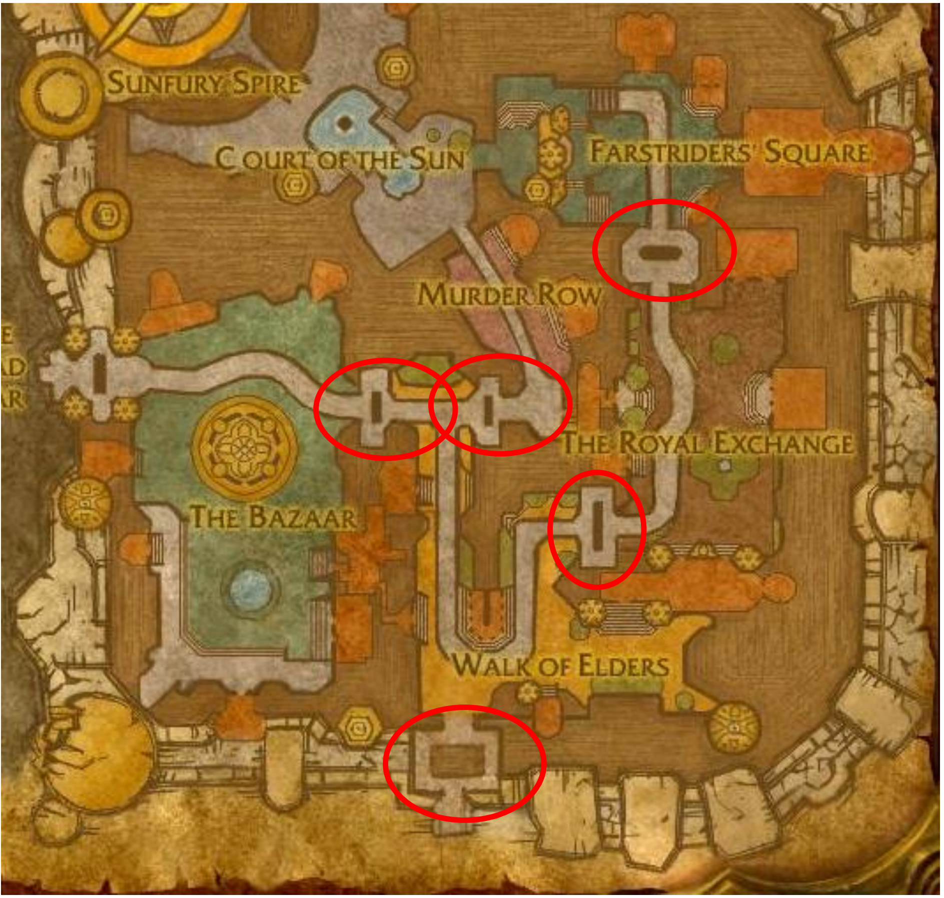
- Use of LoD’s - at the right time and place
- on-the-fly loading/unloading of assets
- Sometimes: *specific* level design
- Optimizations specific for game genre
- More optimizations of CPU/GPU (platform)

Lots(!) of work

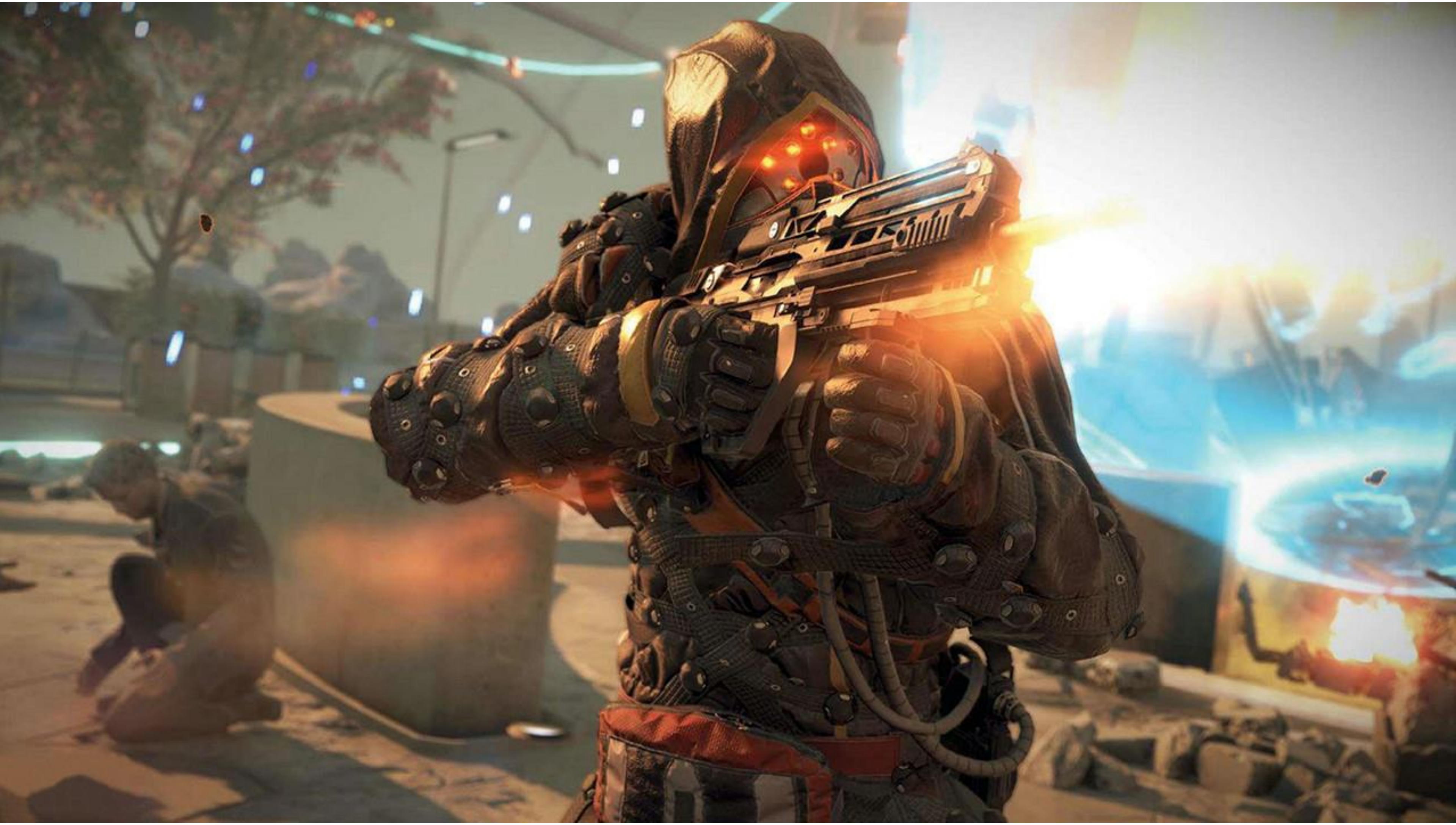
Example ‘level design’

- Simple example: World of Warcraft





AAA example: killzone shadow fall



‘behind the scenes’…

PDF: “Killzone shadow fall Post mortem” (on VLO)

…Reads like a technical journal (from Star Trek!)

…and about 95% of it is *graphics* related

Similar cases...

- GTA V
- Arkham series
- Call of Duty(‘s)
- Battlefield (‘s)
- Assassins Creed (‘s)

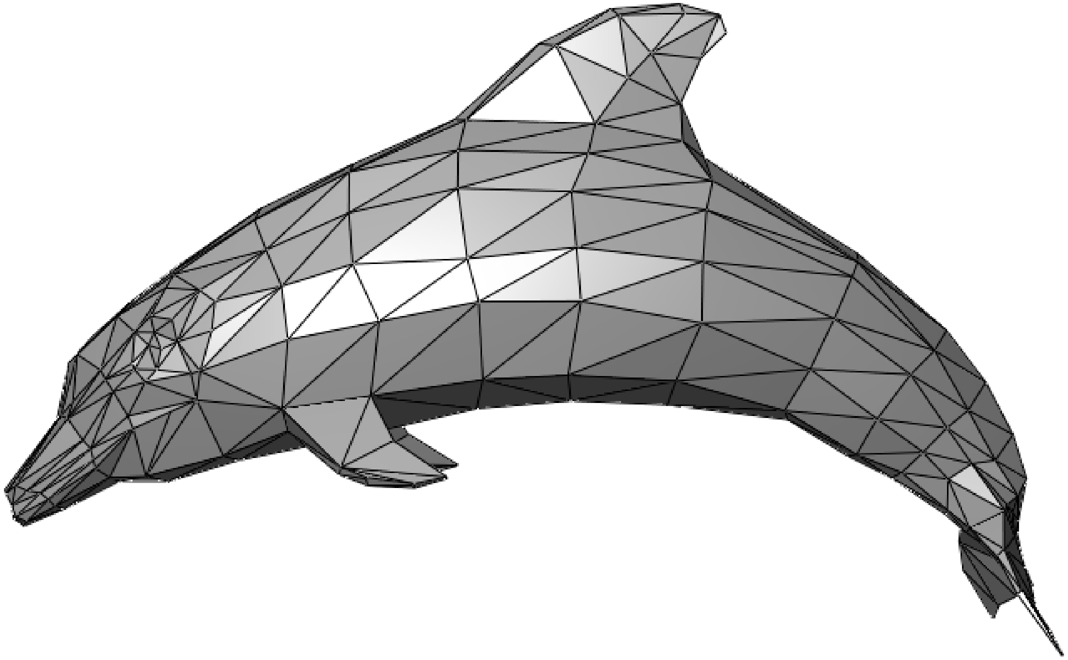
…(you name it)

And further

- Smaller games still need a thorough knowledge base
- Games for phones and tablets

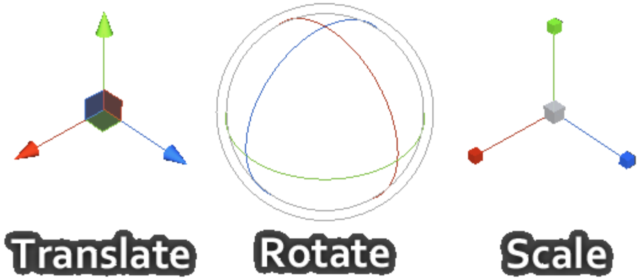
3d data

- Mesh
  - Triangles
  - Vertices
- ..and more...

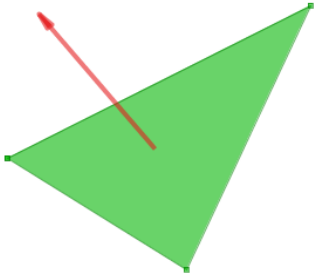


...

- Aside the mesh:
  - 1 Transform (p. mesh)
    - Pos, rot ,scale

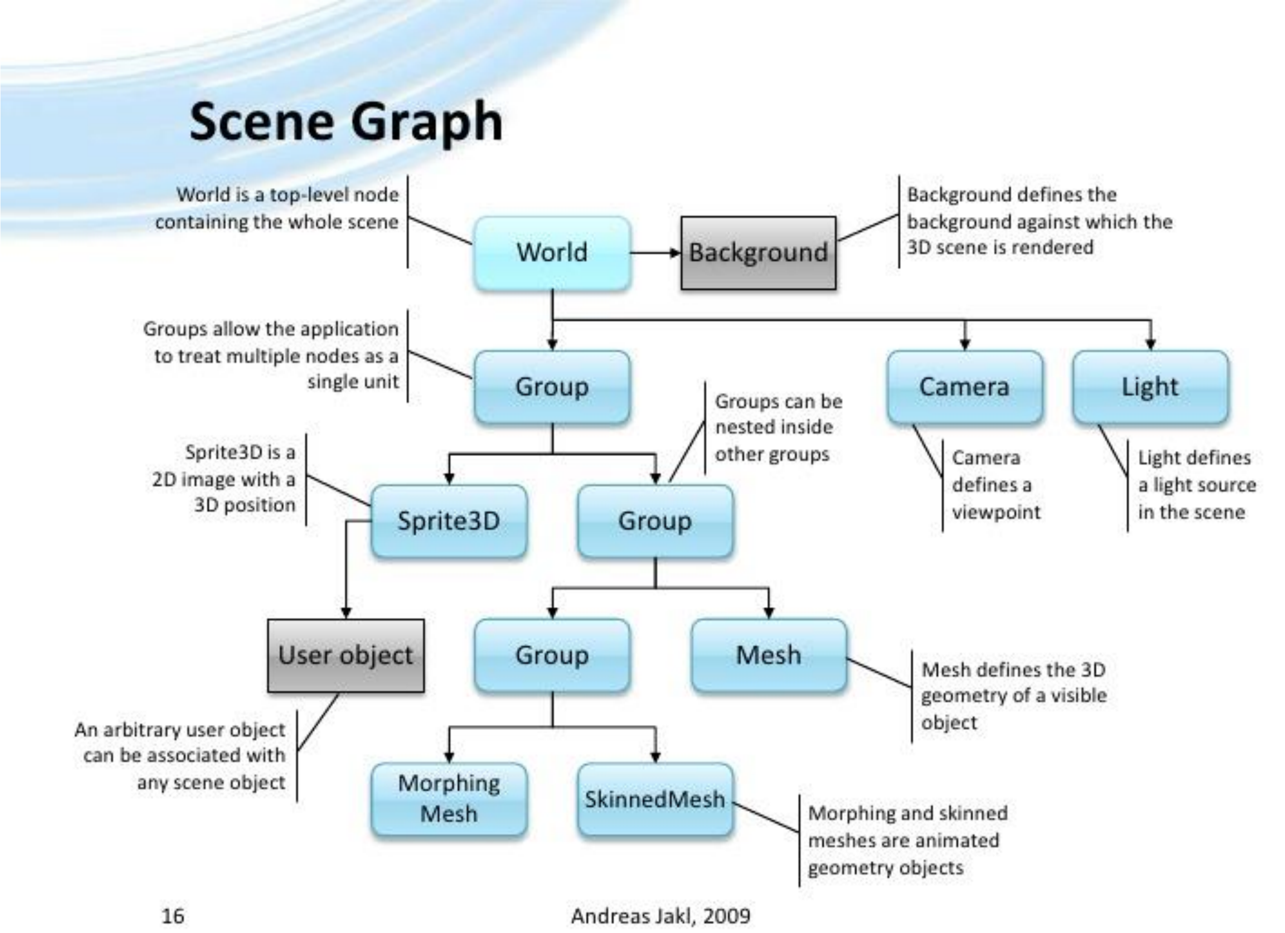


  - UV coords(p. vertex) //kan ook bij les3 - ‘textures’
    - (also color?)



  - Normal p. face
    - -Face Culling-
  - Child Objects…

3d ‘world’ / scene graph



Scene Graph

World is a top-level node containing the whole scene

Background defines the background against which the 3D scene is rendered

Groups allow the application to treat multiple nodes as a single unit

Groups can be nested inside other groups

Camera defines a viewpoint

Light defines a light source in the scene

Sprite3D is a 2D image with a 3D position

Mesh defines the 3D geometry of a visible object

An arbitrary user object can be associated with any scene object

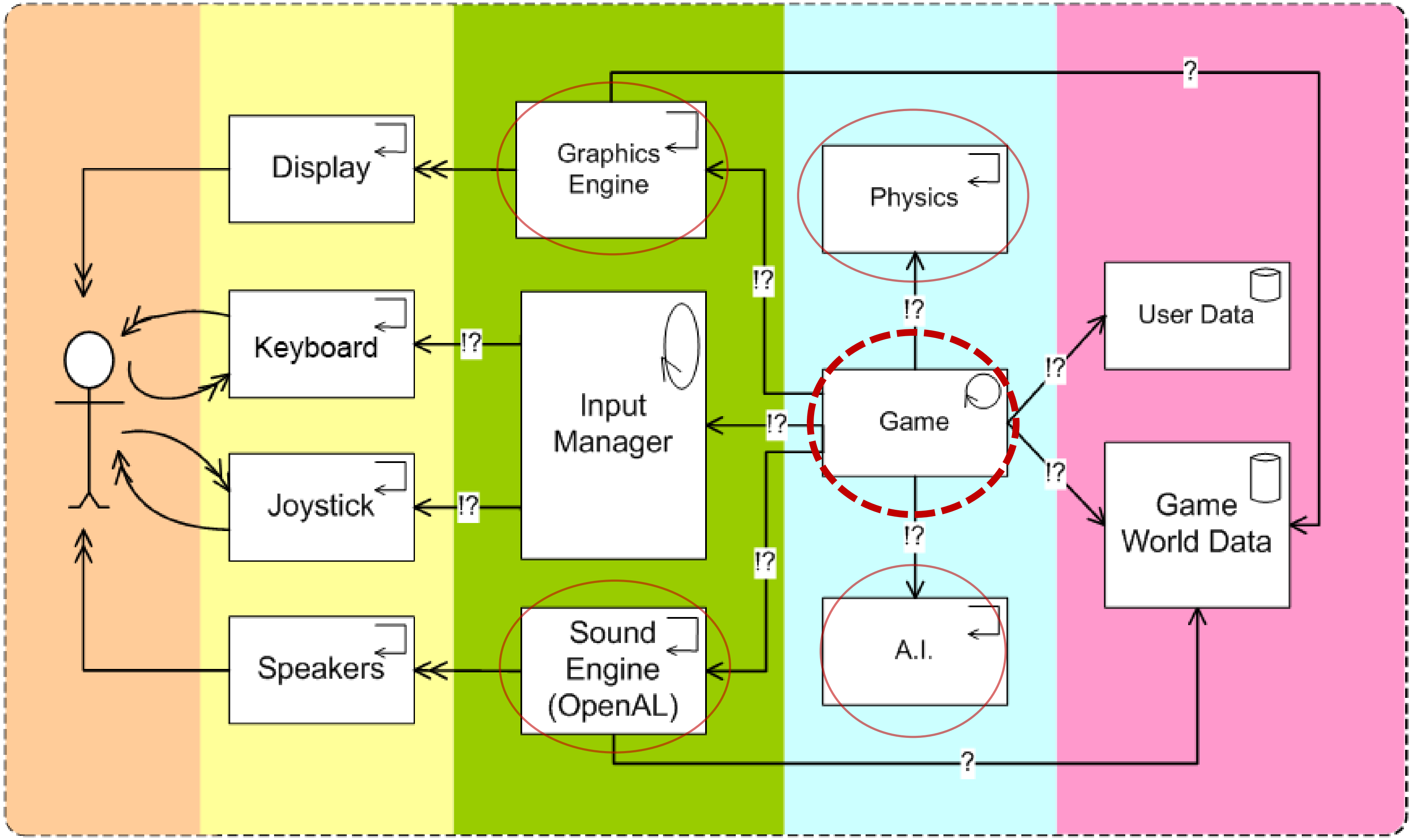
Morphing and skinned meshes are animated geometry objects

16 Andreas Jakl, 2009

Based on the scene graph image from the JSR 184 API documentation

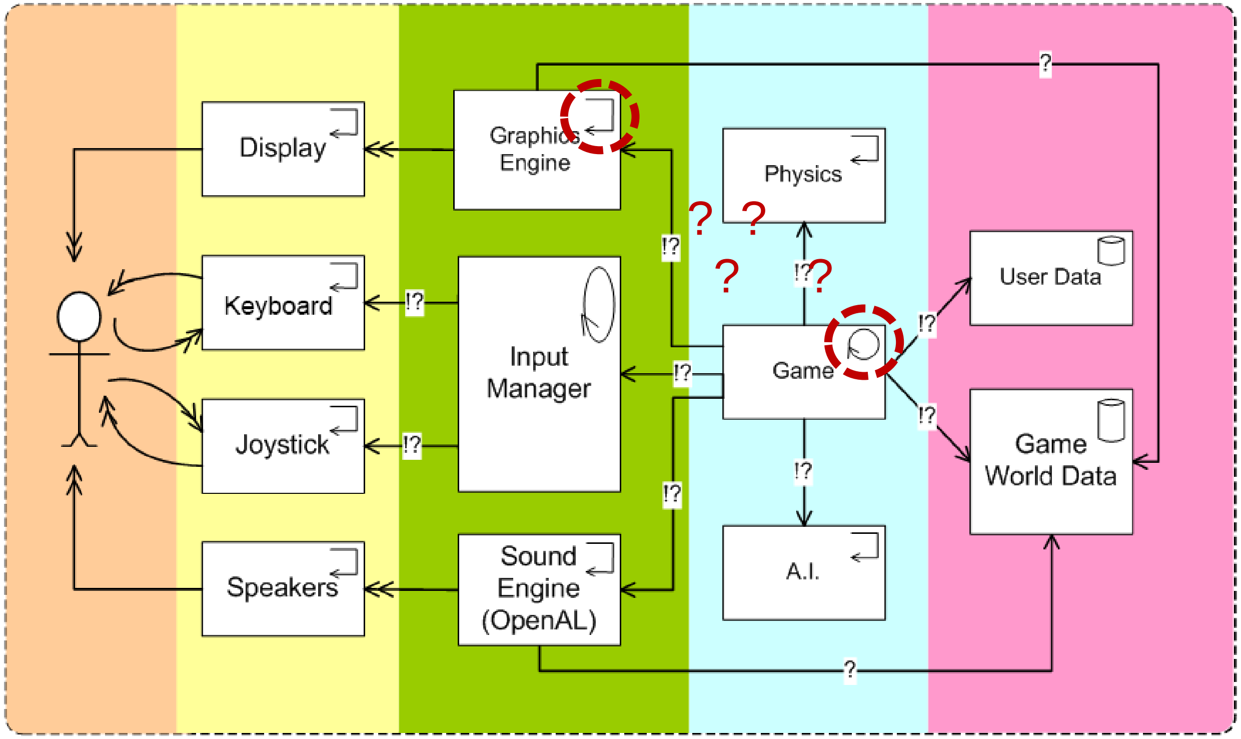
- Example (3d Studio max / Unity 3D)

The Big Picture: Game engine



Frame rate(S)

- Before we talk about “30 / 60 fps” concerning graphics, we have to talk about:
  - ***Render*** frame rate vs ***Game*** frame rate (tick rate)



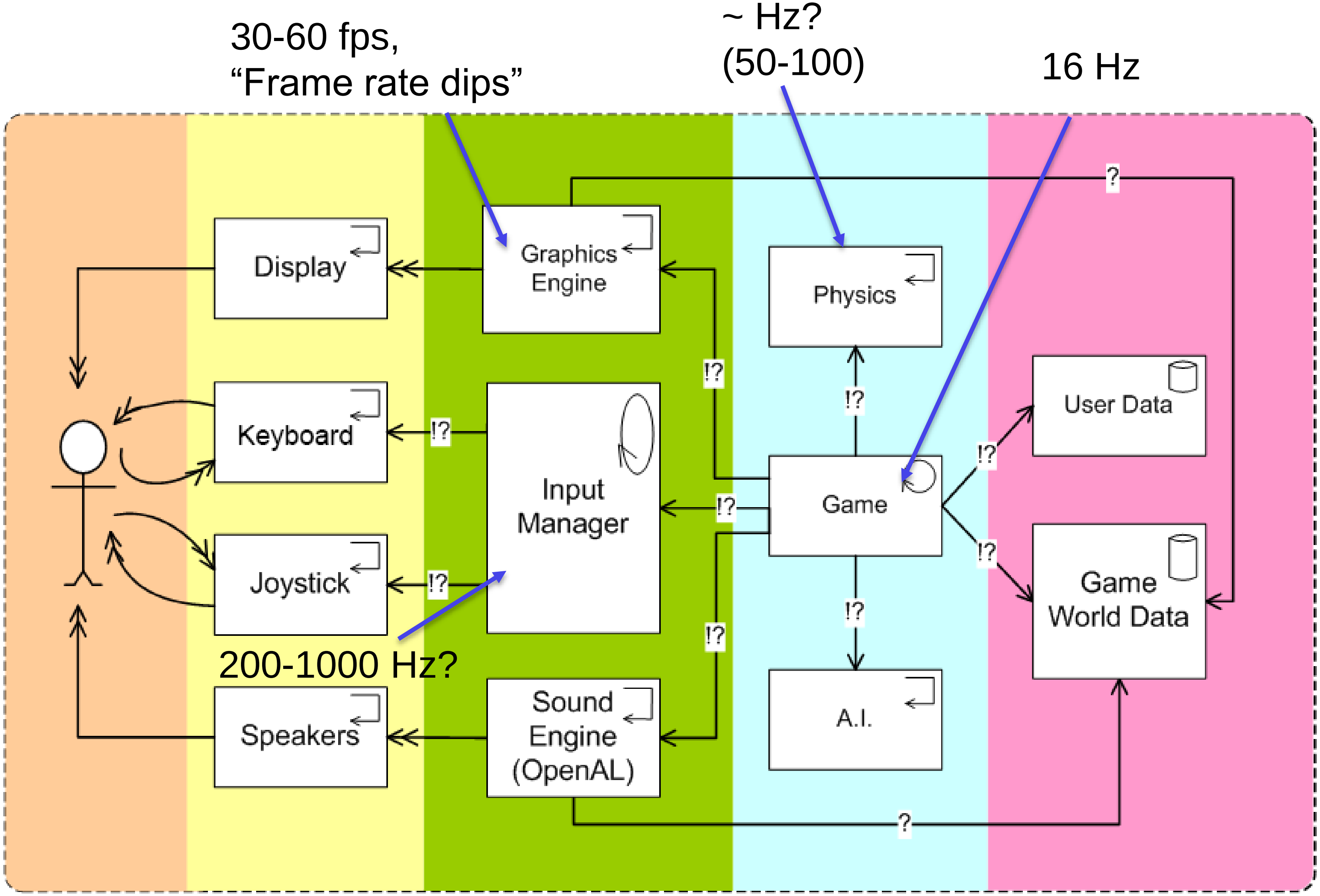
‘fun fact’

- Starcraft 2
  - Graphics fps? depends on hardware… (probably 60 fps)
  - Game ticks? **16 FPS**

Other examples

	Tick Rate
Supreme Commander:	10
Counterstrike: GO	64 & 128 (e-sports)
Team Fortress 2	66
BF3 / BF4	10 (…yikes!!)

Different rates



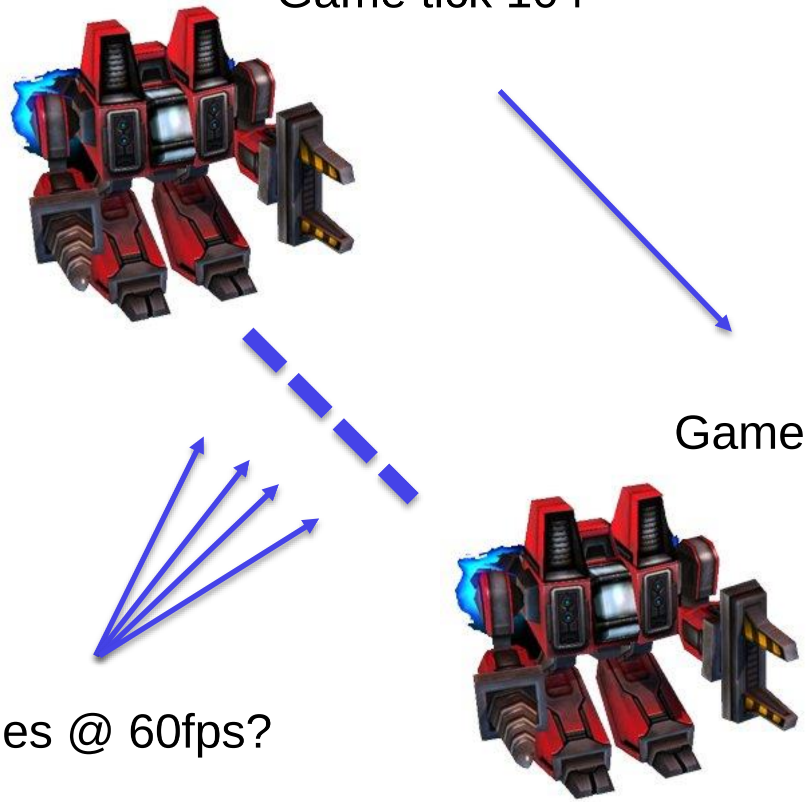
But why?

- Graphics Frame rate is *not* reliable anyway
  - Frame rate ‘dips’ are almost inevitable
  - Different hardware (PC’s) will let fps vary 20-60(-120)

- 60 Hz tickrate with multiplayer games is not easy
  - Network bandwidth
  - Server capacity
    - 100 instances @60 Hz, or 600 instances @10 Hz

- Variable tickrate with multiplayer games is hard
- Physics tickrate, same deal: variable tickrate = disaster

Interpolation!



Game tick 104

Game tick 105

Render frames @ 60fps?

Interpolation

- We know ‘current position’
- We also know its ‘next position’ (pos + deltaPos)

Percentage = time – deltaTime;