Game Technology

Lecture 11 – 16.01.2015
Large Game Worlds and Streaming

Dipl-Inf. Robert Konrad
Dr.-Ing. Florian Mehm

Prof. Dr.-Ing. Ralf Steinmetz
KOM - Multimedia Communications Lab


STEALTH-COM

Today's Games

Typical hardware requirements

- 8 GiB RAM
- 2 GiB Video-RAM
- 50 GiB on disk

All SNES games ever (including all language versions)

- ~3000 games
- ~4.5 GiB

Today‘s Data

One uncompressed texture

- 4096 x 4096 x 4 Bytes = 67108864 Bytes = 64 MiB
- 2 GiB / 64 MiB = 32
- Physically based rendering – typically 4 textures

Killzone 4 CPU data

Sound	553 MB
Havok Scratch	350 MB
Game Heap	318 MB
Various Assets, Entities, etc.	143 MB
Animation	75 MB
Executable + Stack	74 MB
LUA Script	6 MB
Particle Buffer	6 MB
AI Data	6 MB
Physics Meshes	5 MB
Total	**1,536 MB**

Killzone 4 GPU data

TECHNISCHE UNIVERSITÄT DARMSTADT

Non-Steaming Textures	1,321 MB
Render Targets	800 MB
Streaming Pool (1.6 GB of streaming data)	572 MB
Meshes	315 MB
CUE Heap (49x)	32 MB
ES-GS Buffer	16 MB
GS-VS Buffer	16 MB
Total	**3,072 MB**

PNG and JPEG

PNG

- Lossless
- Compression highly dependent on image content

JPEG

- Lossy
- Generally strong compression

Both

- Slow decompression
  - Can slow down loading times
- Not possible to access a single pixel while compressed
  - Not usable for image computations aka not usable as a texture format

Texture Compression

Many different formats

- S3TC, PVRTC, ASTC,…
- Has to be supported by GPU and Graphics API
- Of course much of it is patented and hard to standardize

Design goals

- High compression
- Low visual degradation
- Efficient single pixel access
  - → What we need during fragment shader
  - Constant size of a pixel or a pixel block

Example

Prerendered images for a mobile VR game

Recommendation for Gear VR: 4096x2048

Device: Note 4, 3GB RAM, shared between GPU and CPU

- Uncompressed images: 33,55 MB
- PNG: ~11 MB

Need to minimize data transfer and storage (complex format), need to minimize storage in GPU memory (fast format)

Strategies

2K Texture

- Uncompressed: 4 byte * 2048 * 2048 = 16.77 MB
- DXT1: 2048 * 2048 / 16 * 8 bytes = 2.1 MB
- PNG: ~6 MB (Depends on content and compression details)
- JPEG: ~1 MB (Depends on content and compression details)

Save compressed for GPU (e.g. DXT)

- Quick file load direct into memory
- Fast, simple
- Requires much space

Save in complex format, convert to compressed while loading

- Smaller file sizes (e.g. mobile)
- Longer loading times

Possible compression strategies

Less than 8 bits per color might be ok

The eye‘s color resolution is less then its intensity resolution

Neighboring pixels likely have similar colors

S3TC

Originally developed by S3 Graphics for the Savage 3D graphics accelerator

Also known as DXTn/DXTC (DirectX names)

De-facto standard for OpenGL implementations

Series of 5 algorithms that handle compression differently

- Mainly depending on the way alpha is treated

Block-based compression algorithm

- Input always 4x4 pixel block
- Output 64 or 128 bits

DXT1

TECHNISCHE UNIVERSITÄT DARMSTADT

Input: 4x4 block

Output

- c0: Color encoded with r=5,g=6,b=5 bits (16 bits)
- c1: Color encoded with r=5,g=6,b=5 bits (16 bits)
- 4x4 lookup table with 2 bits per pixel (32 bits)

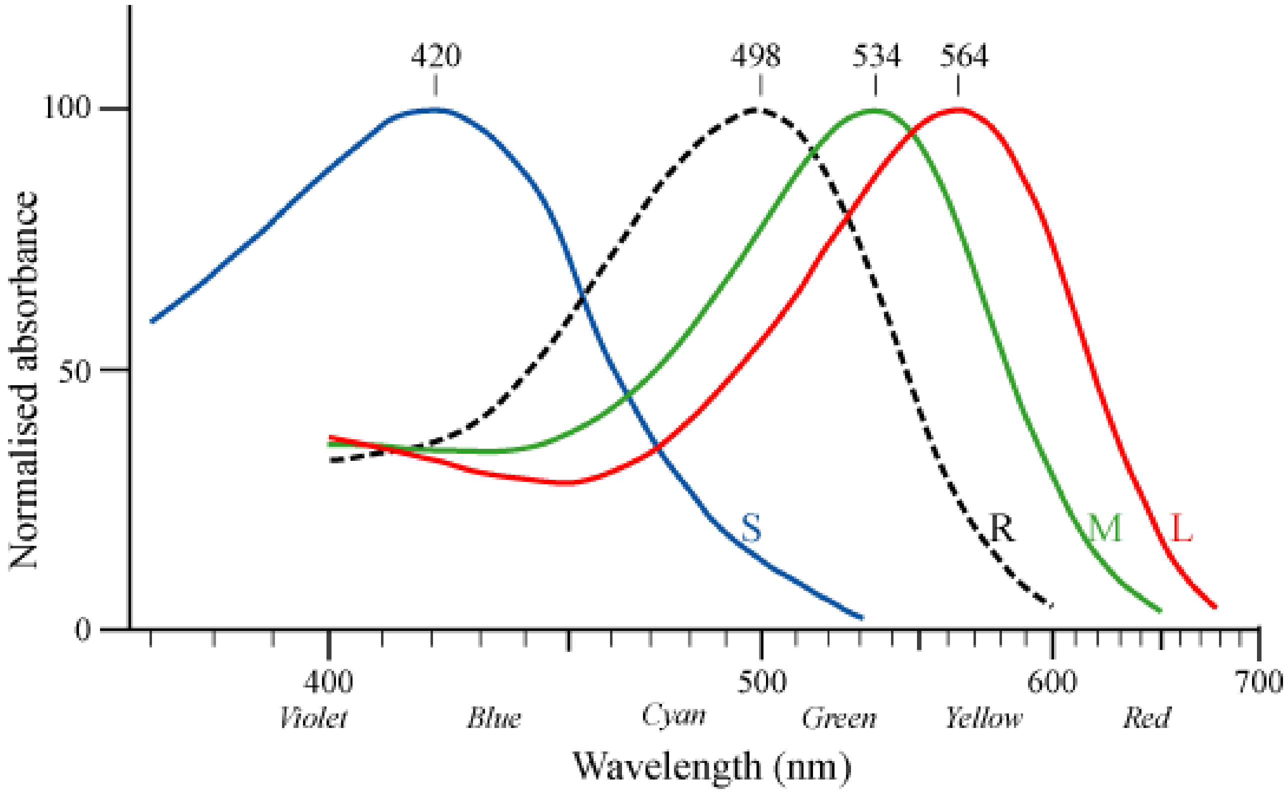
Intermediate values

- if (c0 > c1)
  - c2 = 2/3 c0 + 1/3 c1
  - c3 = 1/3 c0 + 2/3 c2
- if (c0 <= c1)
  - c2 = 1/2 c0 + 1/2 c1
  - c3 = transparent black

Why green?

Dotted line: Absorption of cones, Colored lines: Absorption of rods

Overlap in green area → human eyes can better differentiate variations of green than other colors (& 555 would waste 1 bit...)



https://commons.wikimedia.org/wiki/File:Cone-response.png

DXT1 example

Input block

4x4, 32 bits per pixel

FECB00	B3995D	0073CF	4D5357
B3995D	BDE18A	0073CF	6AADE4
BDE18A	72C7E7	00A1DE	6AADE4
72C7E7	72C7E7	6AADE4	6AADE4

DXT1 example

Choose two colors

Here: Max distance

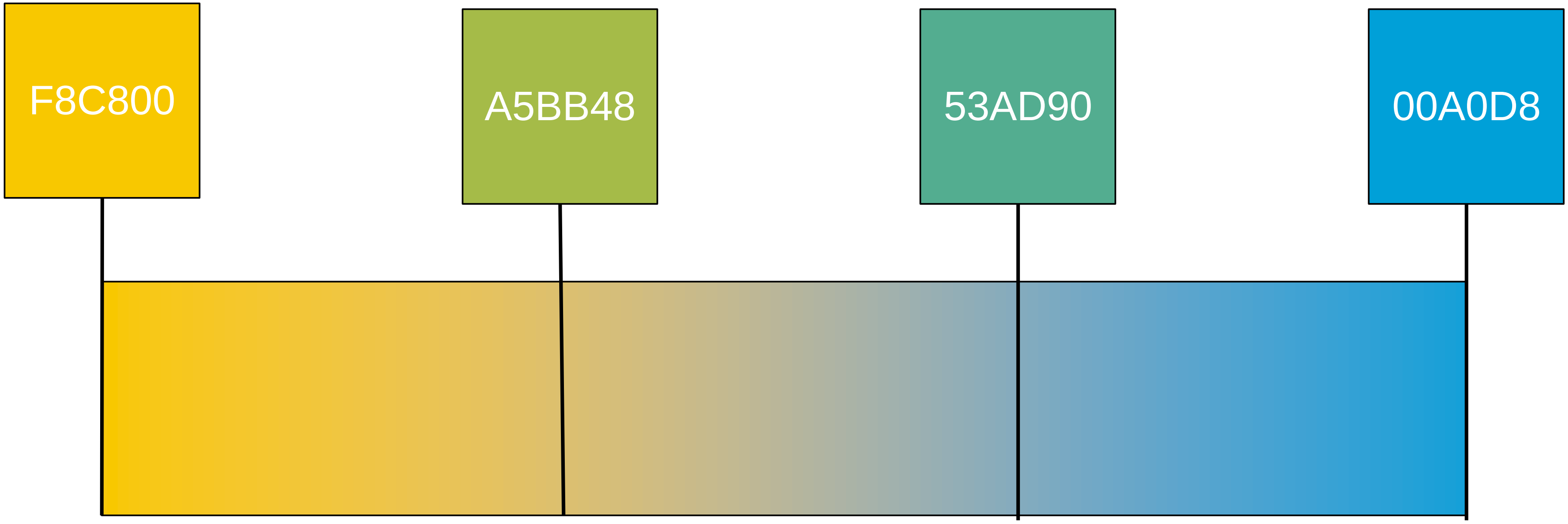
FECB00	B3995D	0073CF	4D5357
B3995D	BDE18A	0073CF	6AADE4
BDE18A	72C7E7	00A1DE	6AADE4
72C7E7	72C7E7	6AADE4	6AADE4

Encode using R5G6B5

FECB00	**Original color 32 bits (here: alpha = FF)**	00A1DE
FE40	**Encoded color 16 bits**	051B
F8C800	**Decoded color 32 bits → quantization error**	00A0D8

Build the palette

Choose c2 and c3 to lie at 1/3 and 2/3 between c0 and c1



Choosing endpoints

Determines the quality of the result

Can apply several strategies

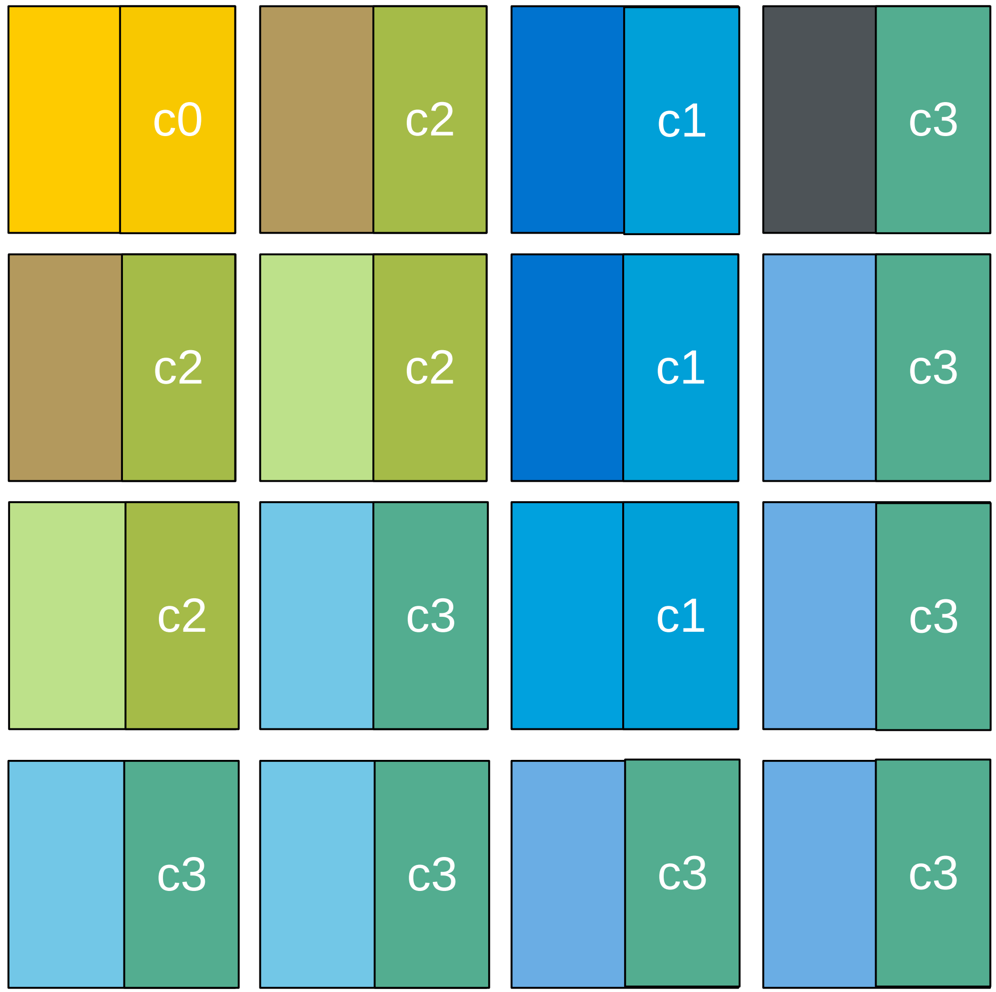
- Local: Only within our block
- Global: Optimize over the image

Principal Component Analysis

"Bounding Box", choose minimum and maximum values along 3 axes

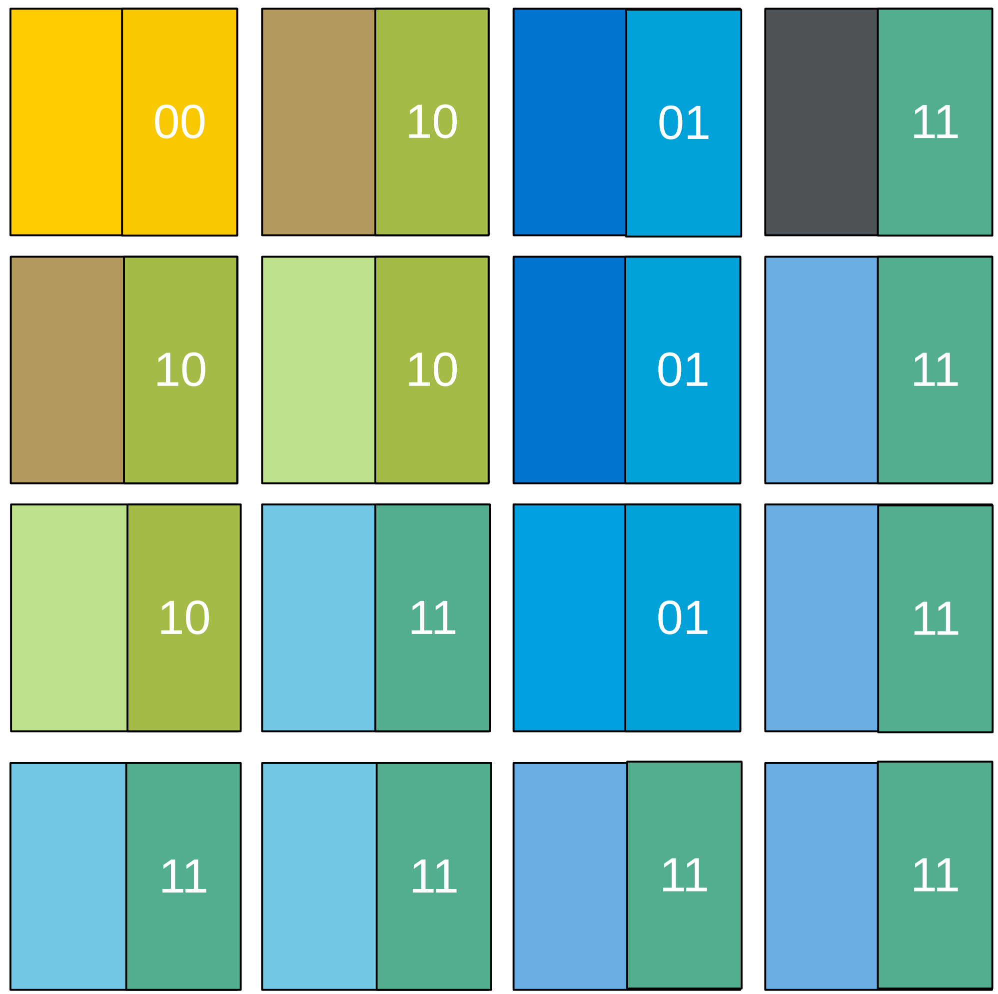
Find the closest colors

Compare to colors in the block to c0 to c3



Find the closest colors

Color index can be encoded in 2 bits



Compressed block

c0 = 0xFE40; // 565 – 16 bits

c1 = 0x051B; // 565 – 16 bits

LookupTable =

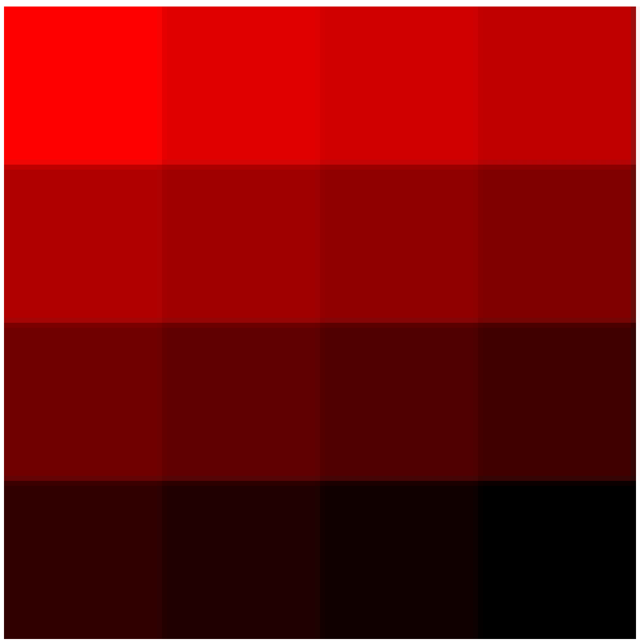
{00, 10, 01, 11, 10, 10, 01, 11, 10, 11, 01, 11}; // 16x2 bits = 32 bits

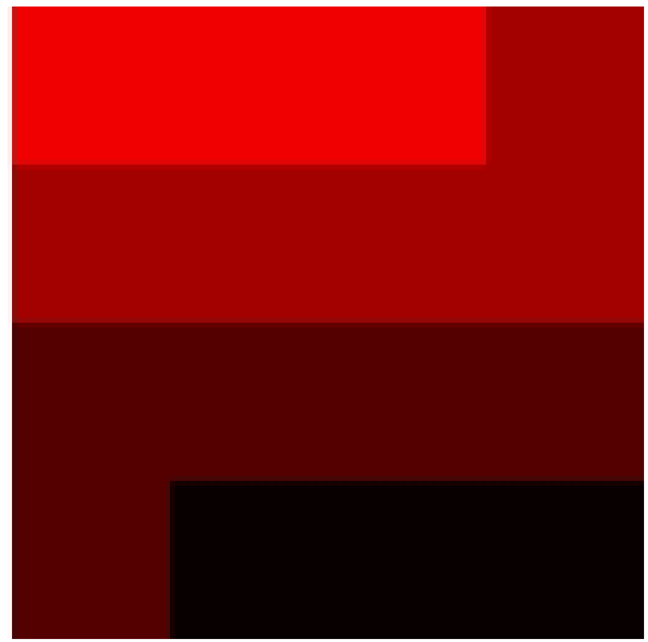
Comparison

DXT1 Examples

Well suited for similar gradients

→ We only lose accuracy due to quantization

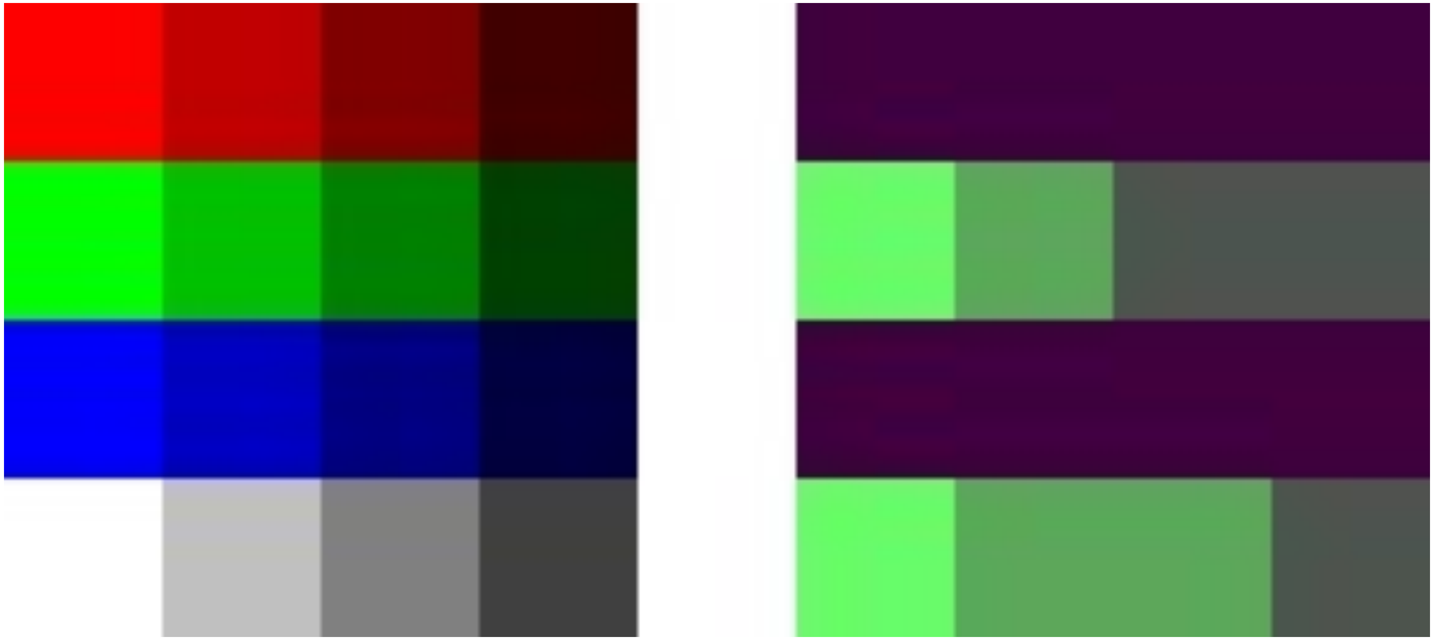




DXT1 Examples

Worst case: Colors not on a gradient

→ We can‘t preserve all colors



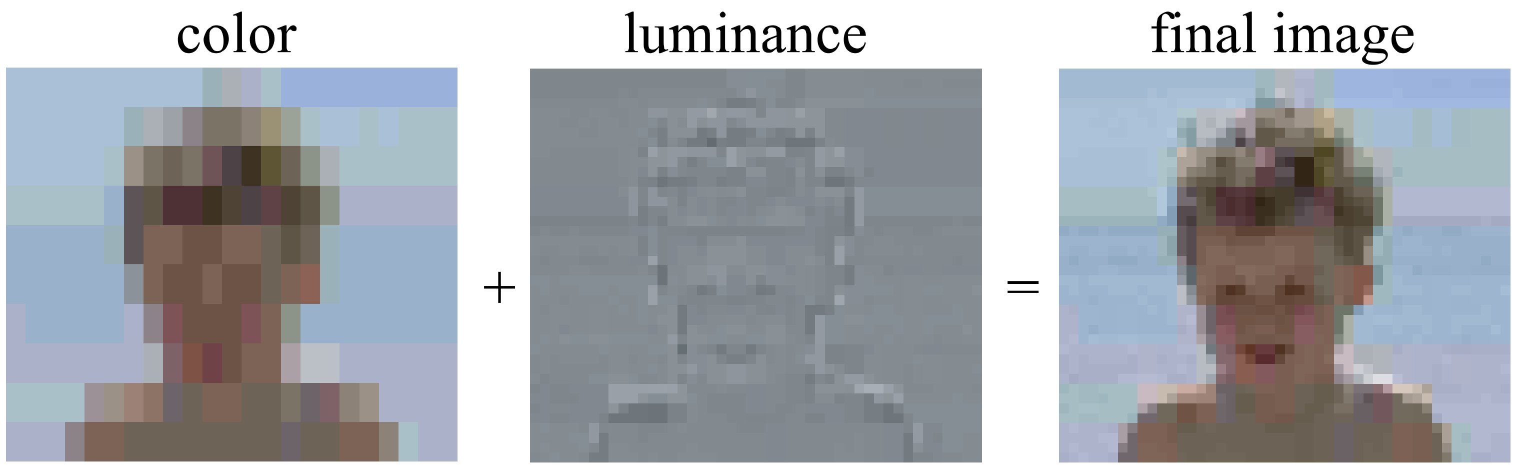
PACKMAN

2x4 blocks

Base color: RGB444 (12 bits)

Luminance modifier lookup index (2 bits per pixel) (16 bits)

Table specifier (4 bits)

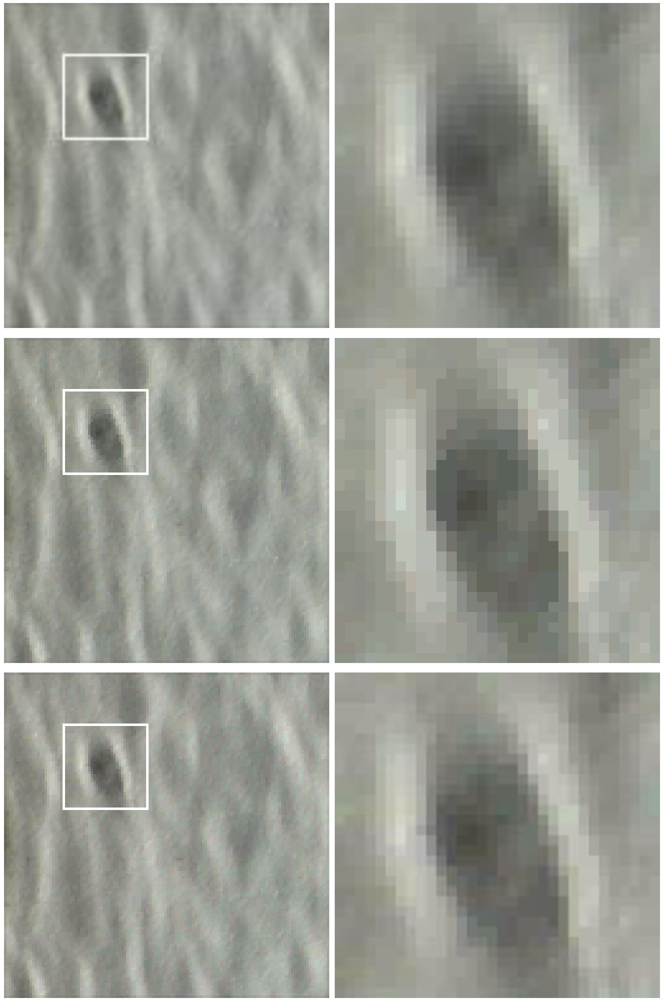


PACKMAN -> iPACKMAN = ETC1

Ericsson Texture Compression

PACKMAN well suited for images with similar luminance

But not well suited for changes in chrominance



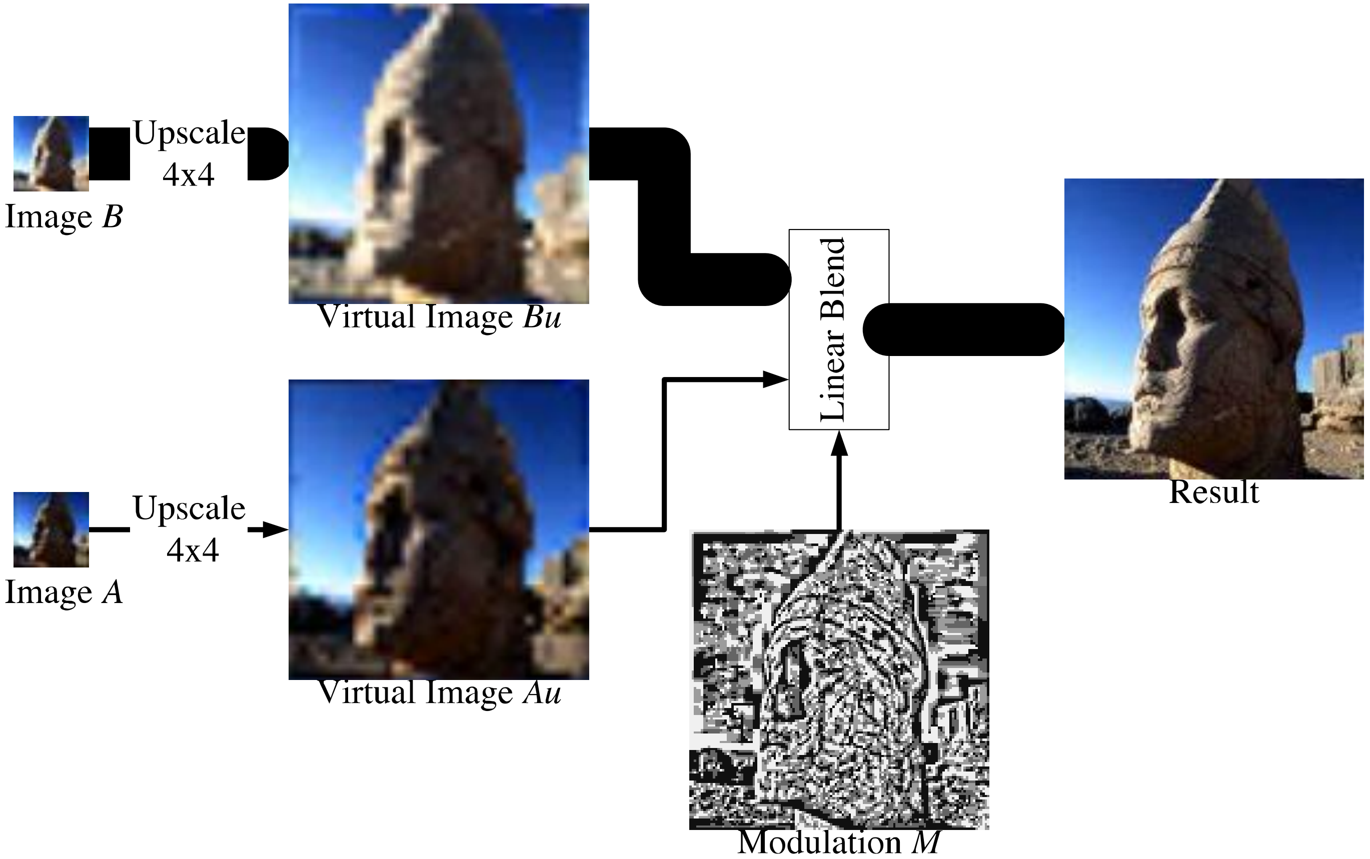
Add an option to combine two 2x4 blocks to one 4x4 block with less compression of chrominance

→ Can adapt to different regions of the image

PVRTC

PVRTC

- PowerVR Texture Compression



Normal Maps, Masks, ...

Compression for images might not be optimal for other textures

- But it might just work
- Swizzling channels can help
- No Alpha used for normal maps
- Some algorithms encode alpha better than other values
  - Move one channel to alpha

3Dc

- $x^2+y^2+z^2=1$
  - $z^2=1-x^2-y^2$
  - One value can be omitted
- Can save normals unnormalized, recover later
- Plus block compression

Manual Compression

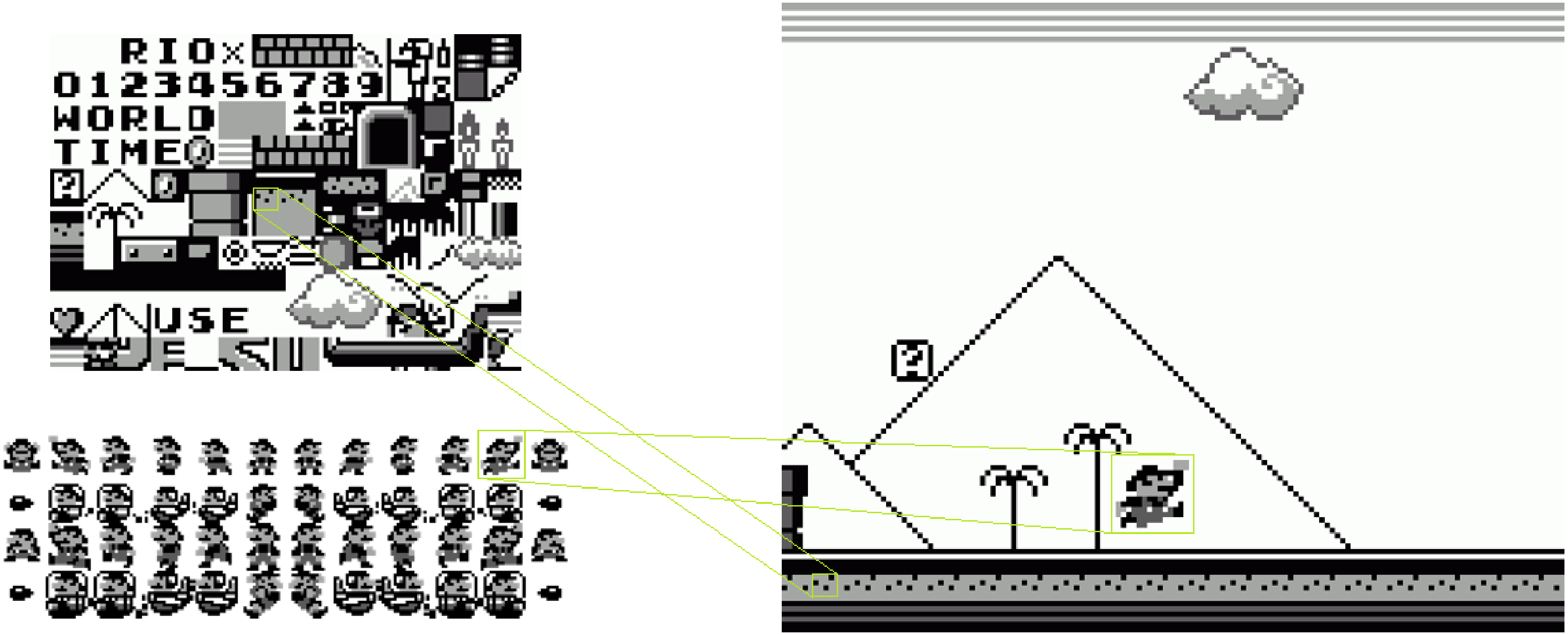
TECHNISCHE UNIVERSITÄT DARMSTADT

Let the artists do the job

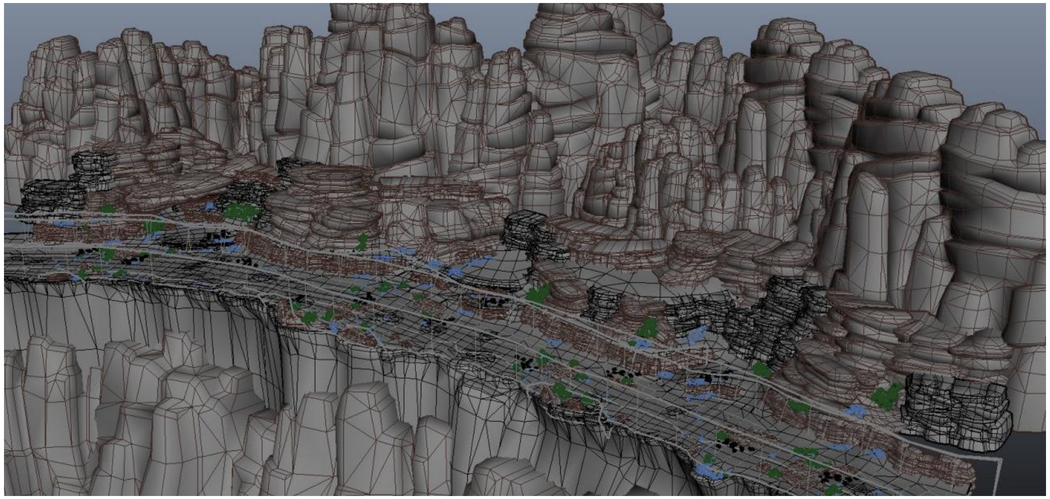
Repeat images over and over

- Nobody might notice it when you do it cleverly

Tilemaps/Tilesets

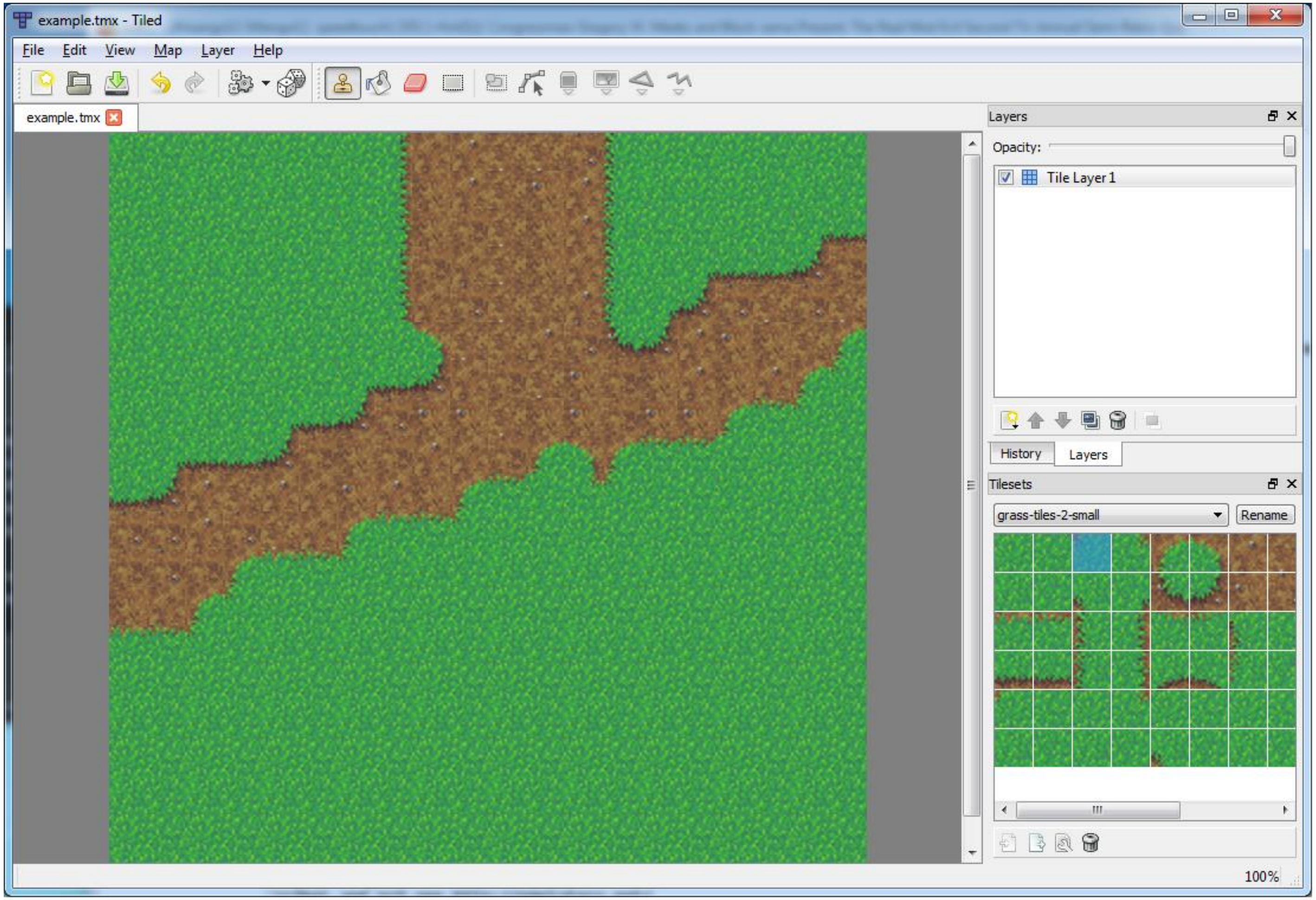


Manual Compression



Uncharted 3, 2011

Tile Editors



http://www.mapeditor.org/

Pitfall: The Mayan Adventure (1994)

Warcraft 3 (2002)

Tile textures in 3D

Used when a specific art look is wanted

Bilinear Filtering

- Would have to use texels from two tiles at tile boundaries
- Complicated
- Expensive, Rarely used

Figure 1: Default Unity terrain

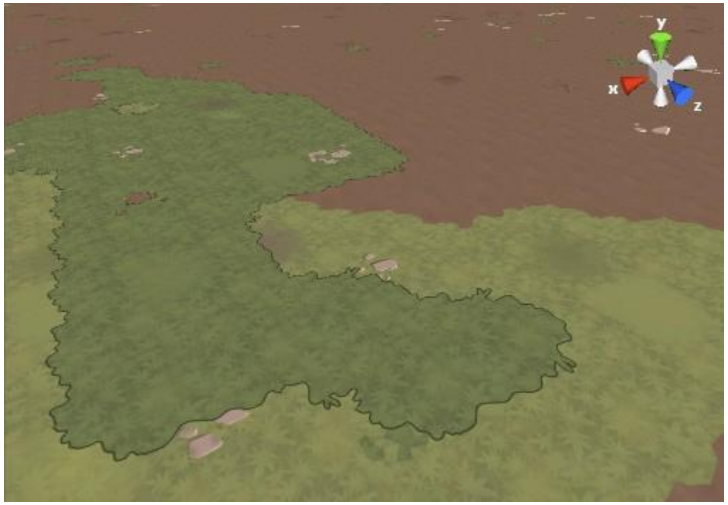


Figure 2: Tiled terrain

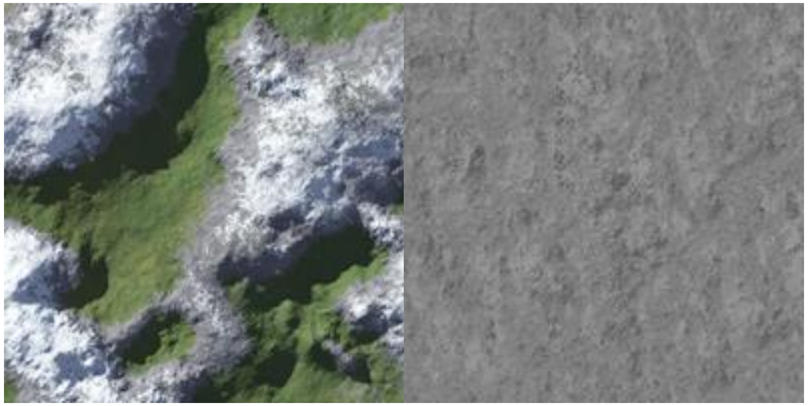
Multitexturing

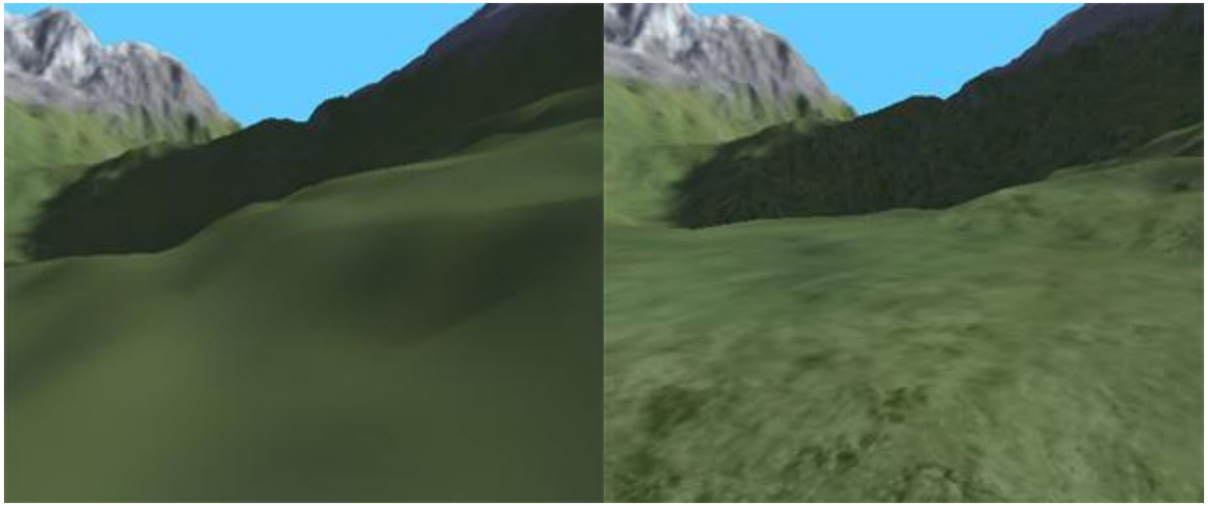
Multitexturing

Multitexturing

Multitexturing

- Broad colors in a base layer
- Details in a repeated texture
- Add details, especially close to the player





http://www.chadvernon.com/blog/resources/directx9/detailing-a-terrain-with-multitexturing/

Decals

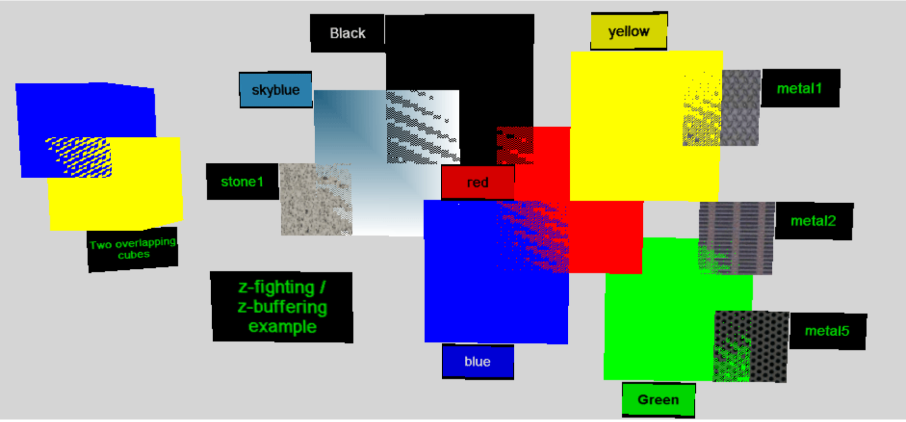
Used to apply textures to surfaces

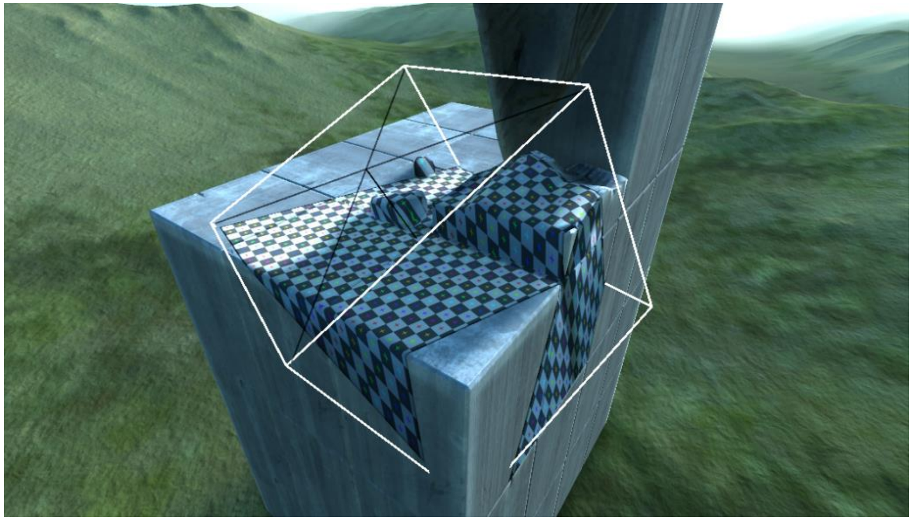
For temporary/dynamic changes, such as bullet holes, ...

Simple implementation: Quad with alpha mask

- Watch out for z-fighting

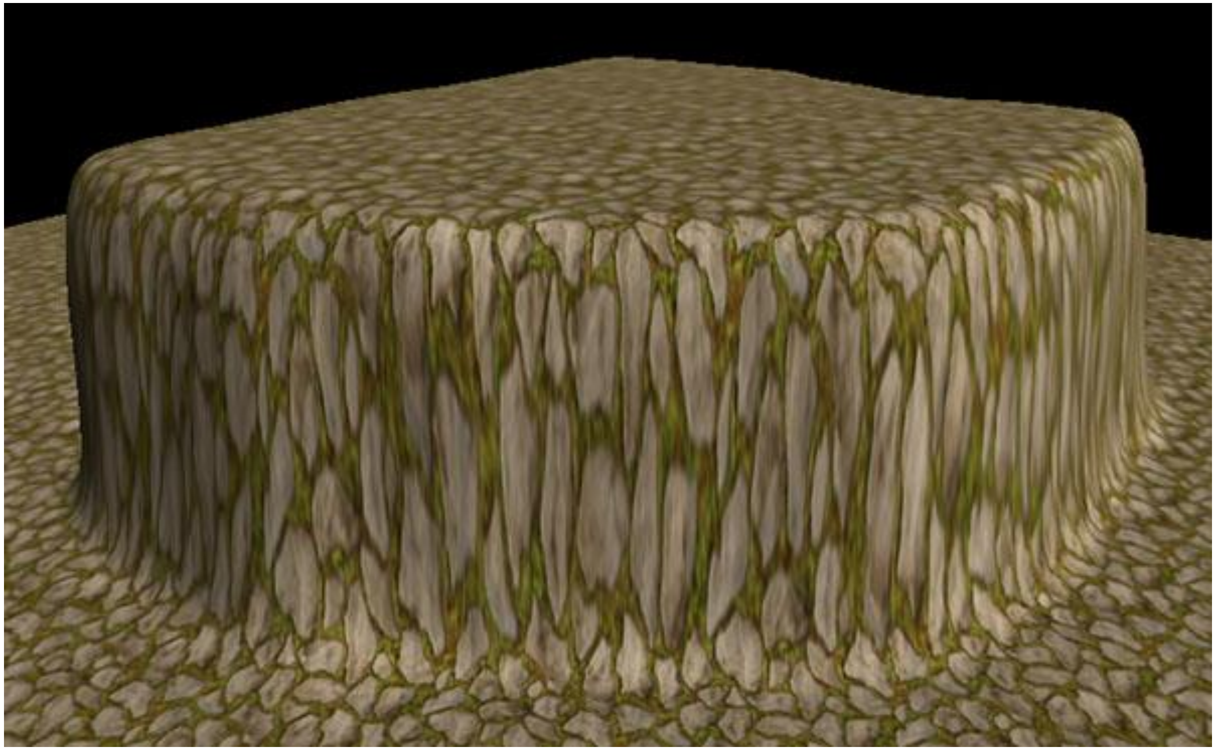
Complex implementation: Projection onto a surface





Terrain Texturing

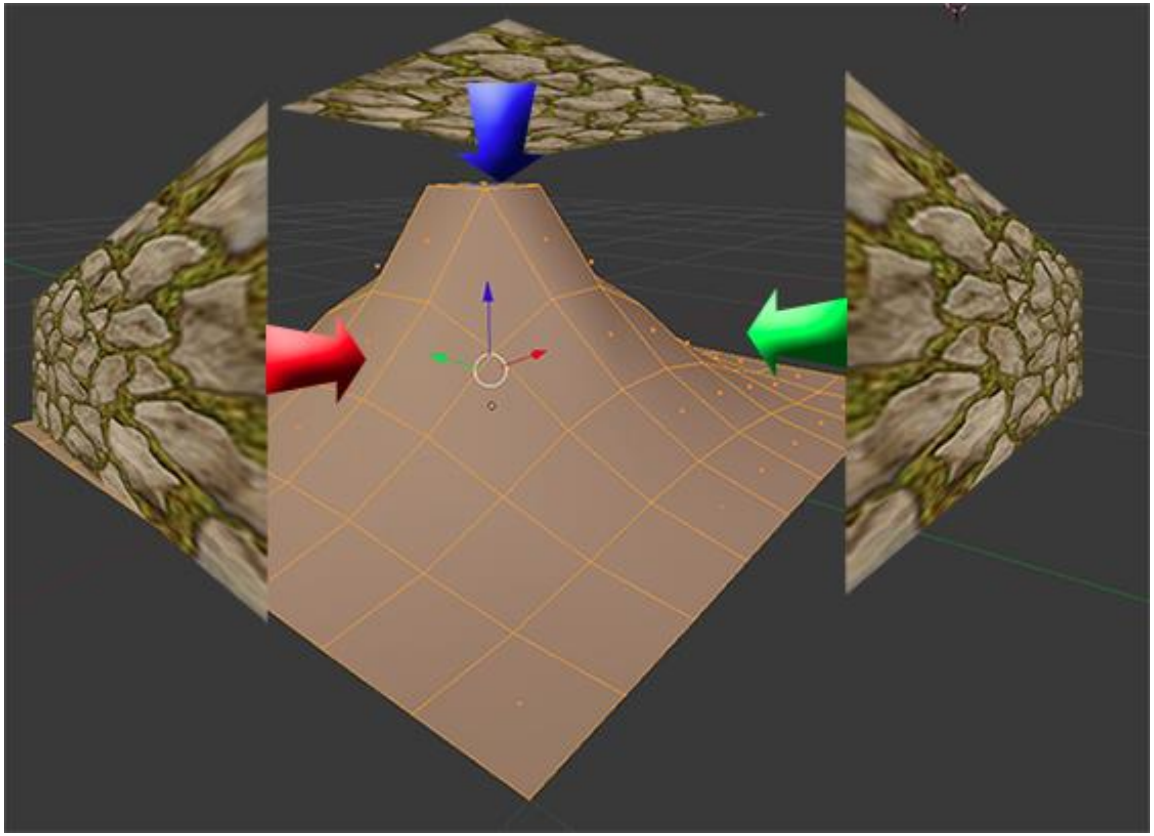
- Terrain texturing is often a 2D operation applied to a 3D surface
- Good for flat (but boring) terrain
- Bad for steep cliffs



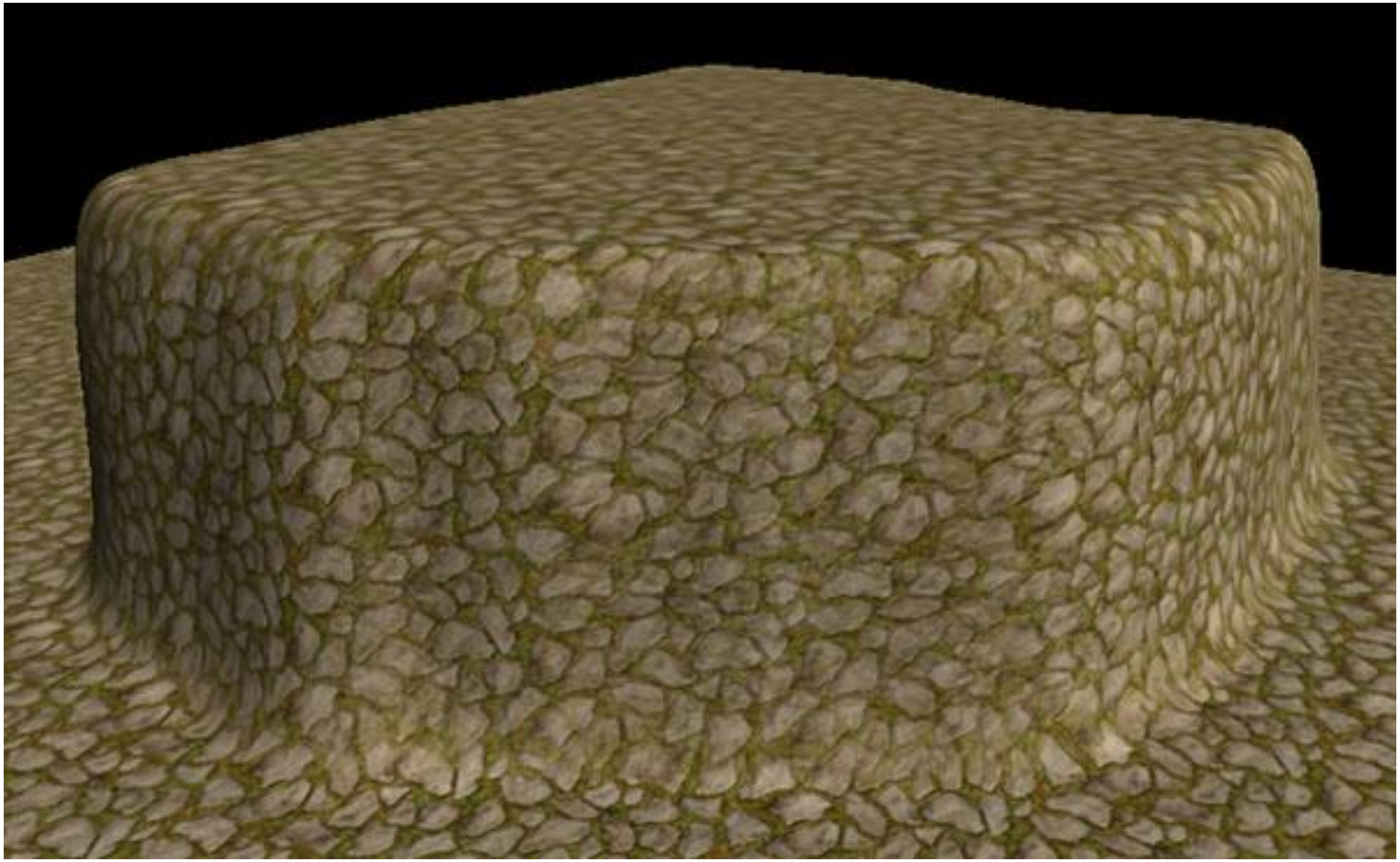
Triplanar Shading

Project the texture onto the geometry from all three axes

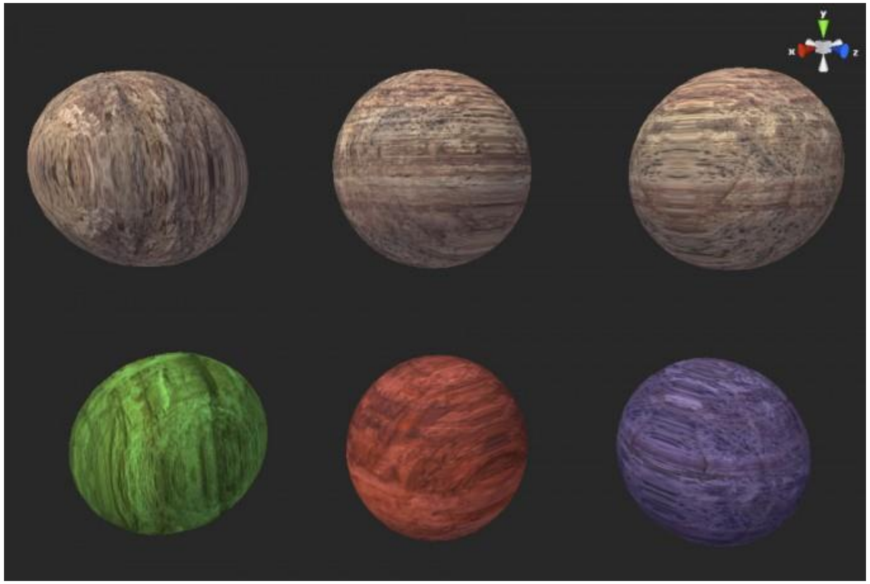
Uses no UVs, but world coordinates

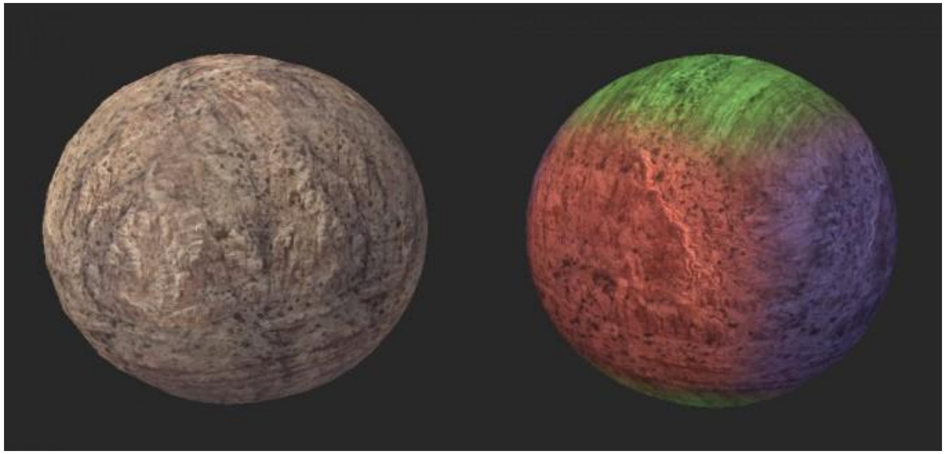


Triplanar Shading Result



Triplanar Shading Visualization





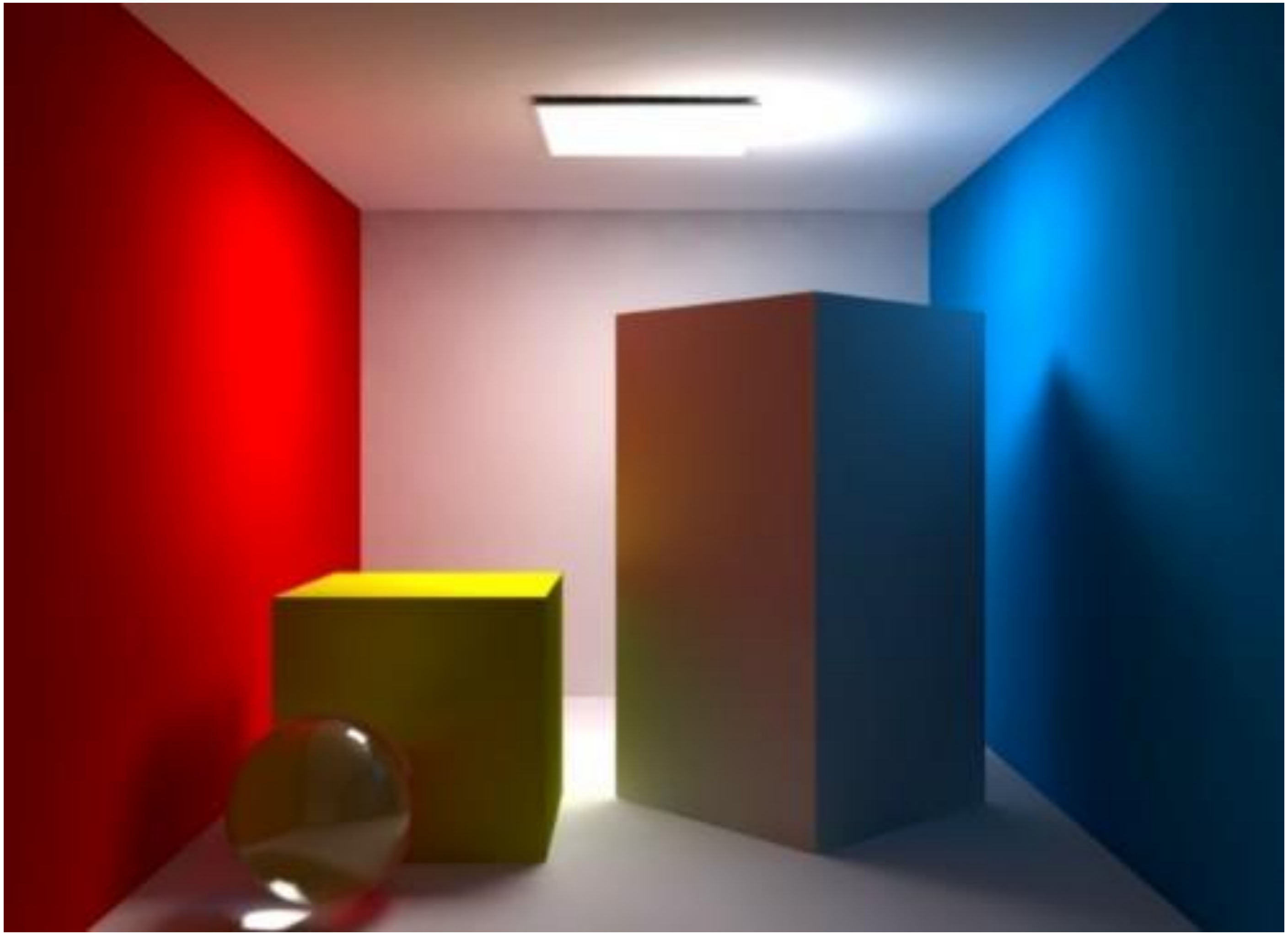
Example: Outcast Reboot HD

Terrain built from 3D tiles (voxels in the original)

Textures applied by triplanar shading

https://www.youtube.com/watch?v=RIsZ_y66HTI

Good lighting can hide a lack of details



Problems

Performance

- More textures, less performance
- Precalculating which polys actually use more textures can help

Needs good tool support

- Scary communication with artists

Streaming

Coarse Streaming

- Load and replace complete assets

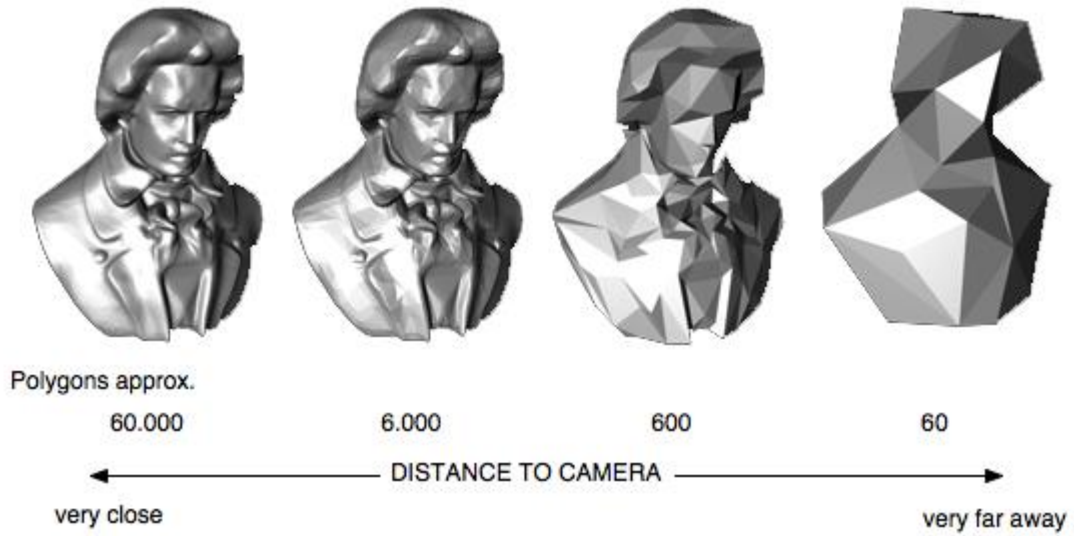
Fine Grained Streaming

- Load and show/play a single asset bit by bit

Coarse Streaming

Similar to level of detail systems

- Load big textures for near objects
- Kick out big textures for far away objects
- Maybe blend texture changes in and out



http://www.opensg.org/htdocs/doc-1.8/lod.png

Problems

Disks are slow and unreliable

- No timing guarantees at all
- Load textures in a second thread, always have an emergency strategy ready (keep super low resolution textures of everything in RAM)

Changing textures at runtime is problematic

- Driver might decide to convert the texture
- Easier on console
- Probably easier with Direct3D 12

Fine grained texture streaming

Rage (2011)

MegaTextures

Really huge textures

- Rage supports textures of up to 128000×128000
  - That‘s ~60 GiB

Compression

- Texture is highly compressed on disk
  - Using lossy JPEG like compression

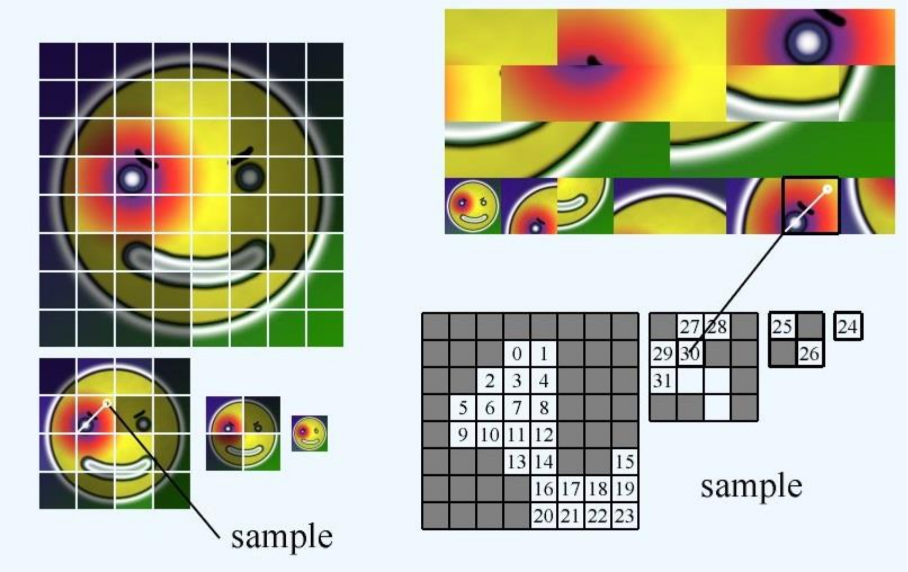
One texture for everything

- Complete world in one texture
- No restrictions for artists
  - But toolsets provide classical multitexturing tricks
  - Artists don‘t manually paint 128000x128000 pixels

MegaTextures Implementation

Similar to virtual memory

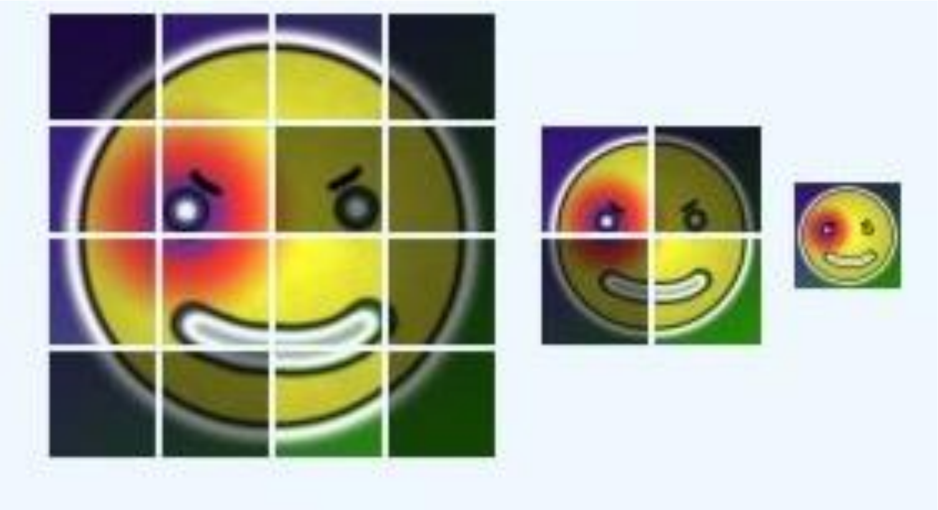
- Application (= pixel shader) believes that there is a huge, continuous area of memory (= texture) it can work on
- Operating system (= texture manager) provides required memory pages by mapping them



For details, see GDC Talk by Sean Barret: https://www.youtube.com/watch?v=MejJL87yNgI

Level of Detail

- Similar to mip maps
- We provide different resolutions of the MegaTexture
- Smaller resolution version should encompass everything we need to sample



MegaTextures

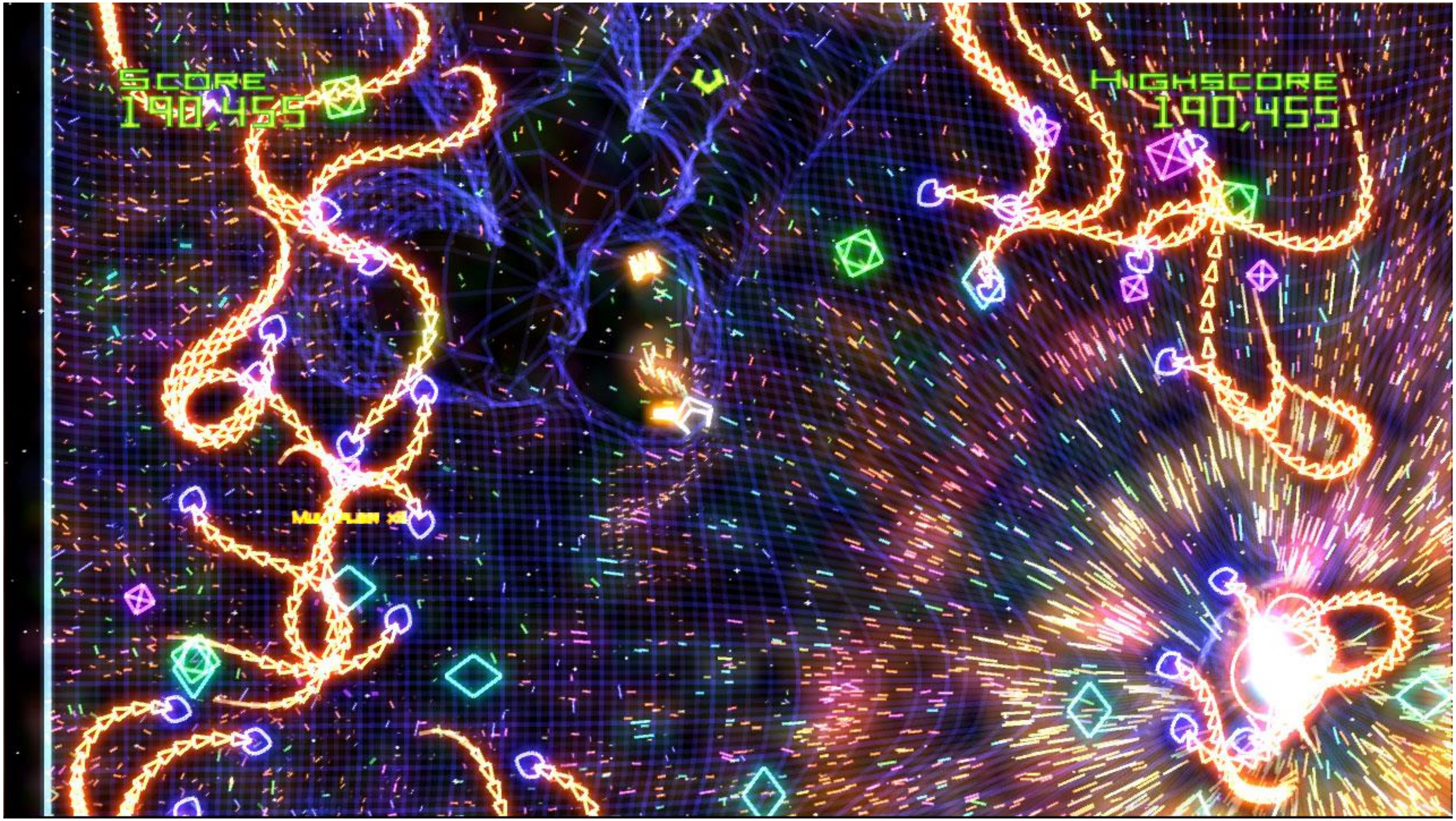
Geometry is split up in tiles

- Engine determines screen size of visible tiles
- Loads texture parts in varying sizes to optimize current view

MegaTextures

https://www.youtube.com/watch?v=YNilHiBpVic

Geometry



Geometry Wars, 2003

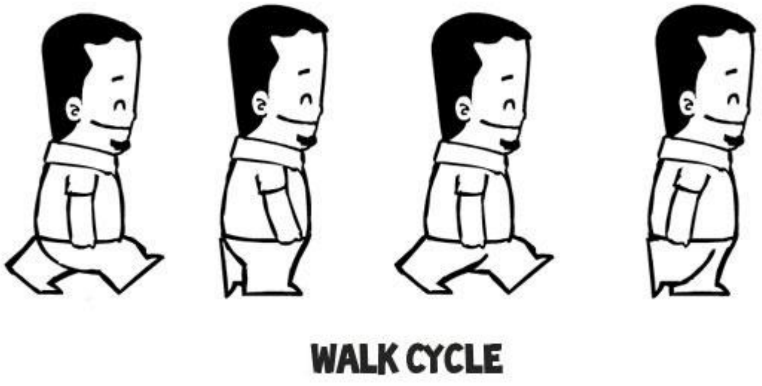
Geometry Compression

Not widely used

No hardware support

Special strategies for animations

- Like skeletal animations, which are tiny
- Replace keyframe blending by raw key frames at large distances

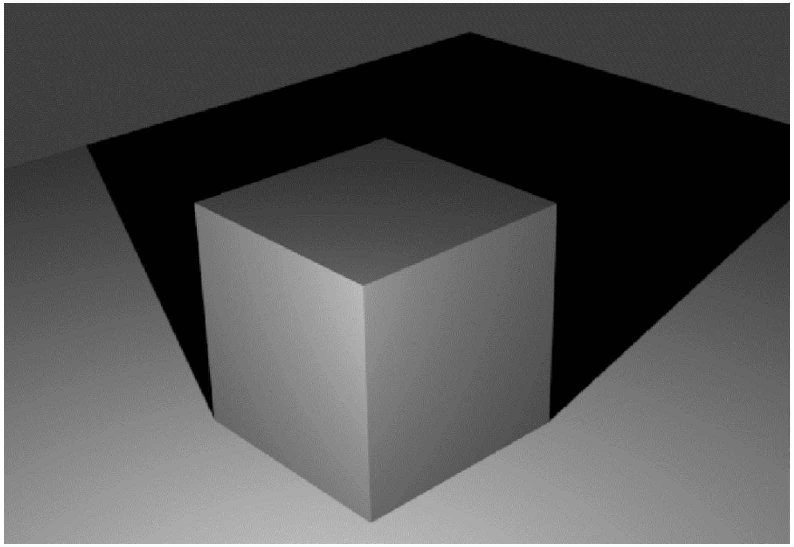
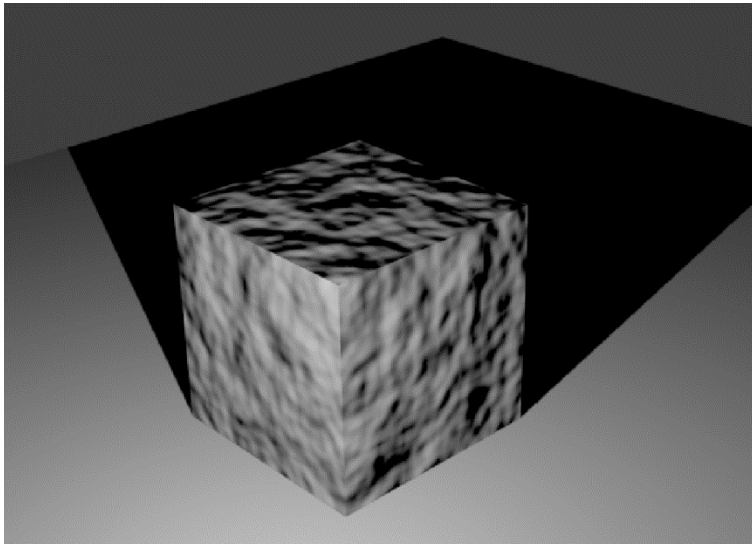


Normal Maps

Remove super detailed geometry

Replace with normal maps

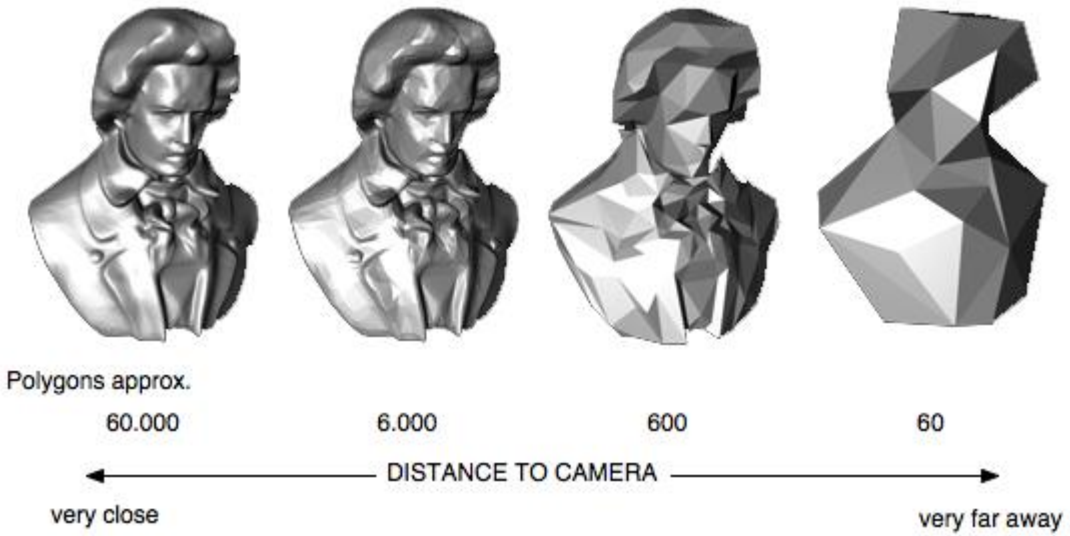
- Which is a form of compression by itself
- Plus normal can be further compressed

Coarse Geometry Streaming

Same strategies as for textures

- Could be directly plugged into a level of detail system



The future – Fine-grained geometry streaming?

Doom (planned for 2016)

id Tech 6

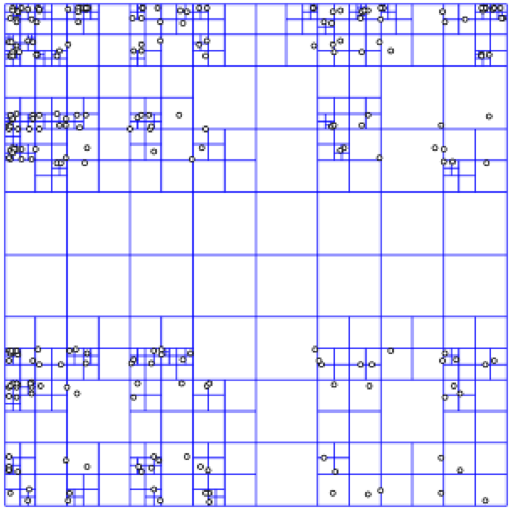
Sparse Voxel Octrees

Voxels

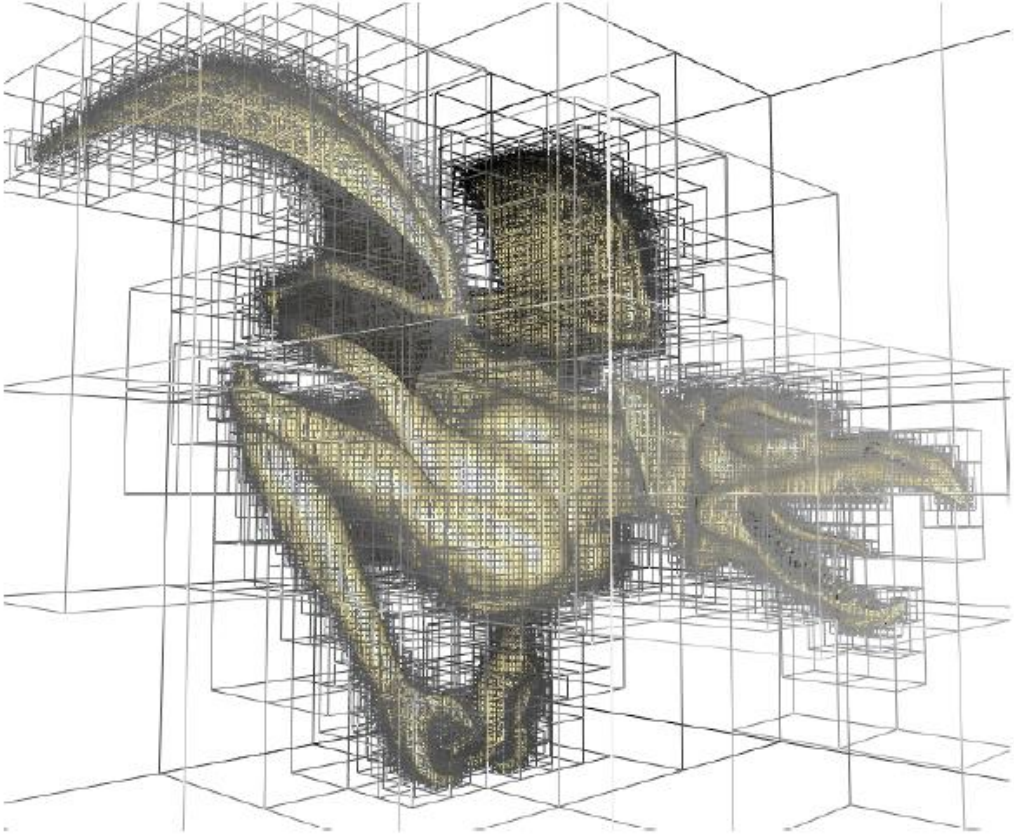
- 3D Blocks
- Can raycast/raytrace relatively efficiently

Octrees

- Subdivide 3D space regularly into 8 sub-spaces



Sparse Voxel Octree

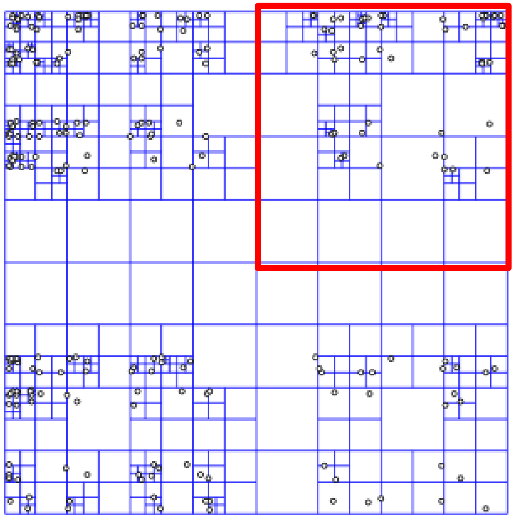


Sparse Voxel Octree

Encode voxelized environment/meshes as octrees

Many cells will be empty → sparse

Cell in an octree contains all children

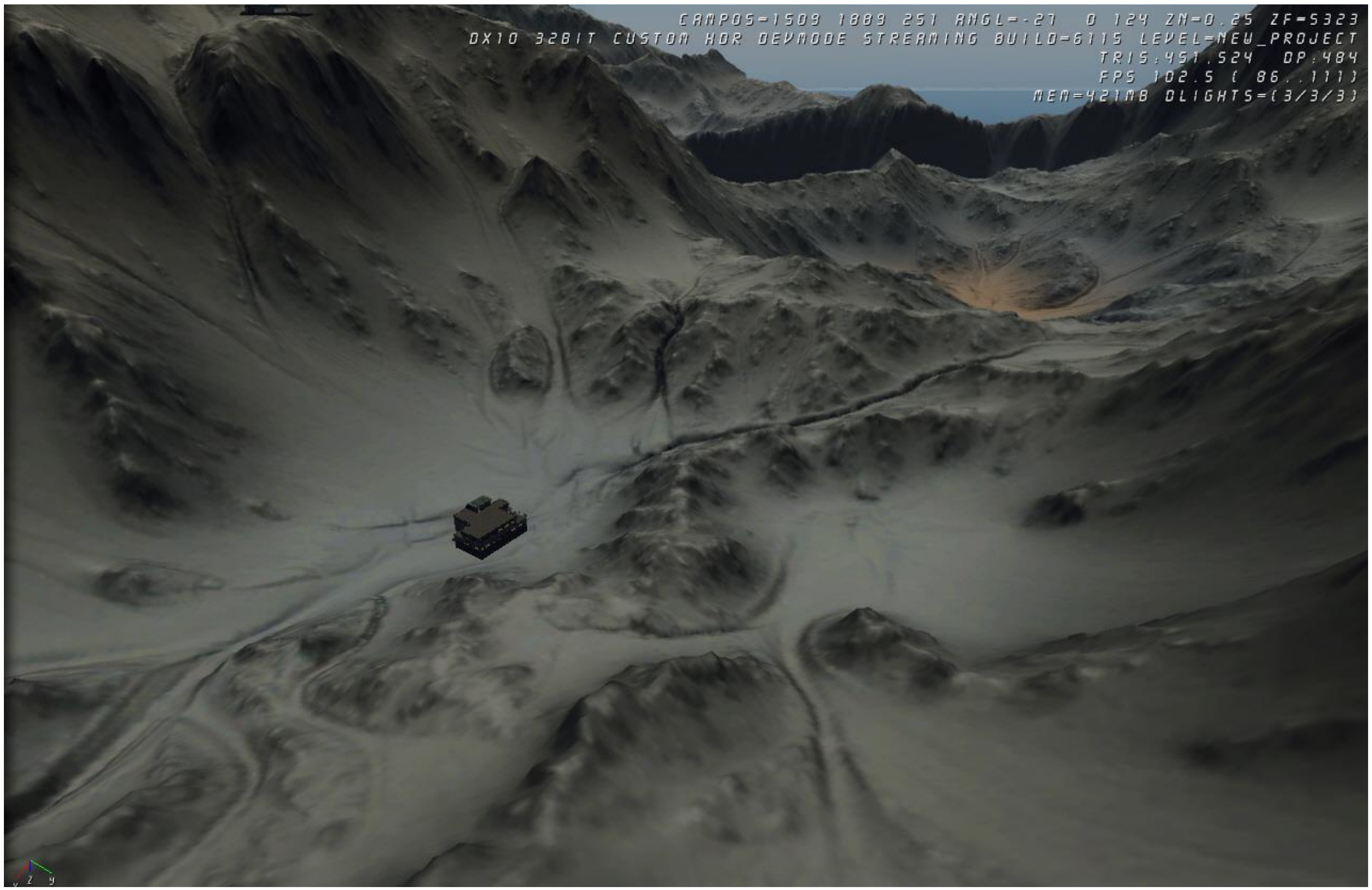


Sparse Voxel Octree

https://www.youtube.com/watch?v=km0DpZUgvbg

Height Maps

Just Y instead of X/Y/Z



Terrain algorithms

Not interested in generation here

- Can use perlin noise, loads of algorithms

Want to handle the terrain efficiently

Naive approach

- Create one vertex at each height map location
- Scale to the size of the terrain

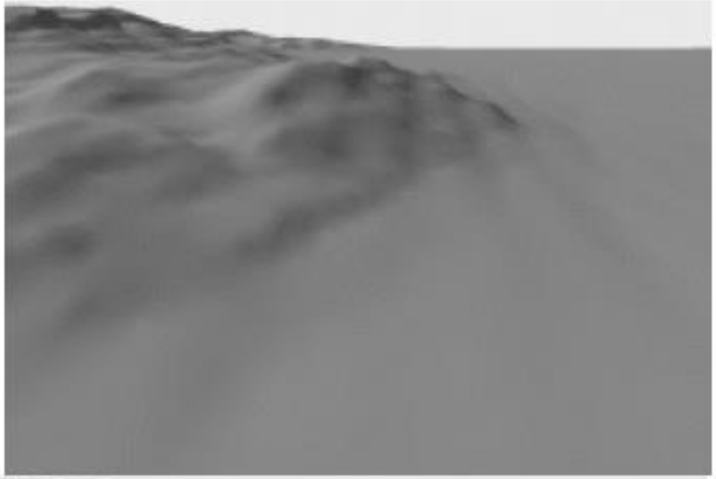
Problems

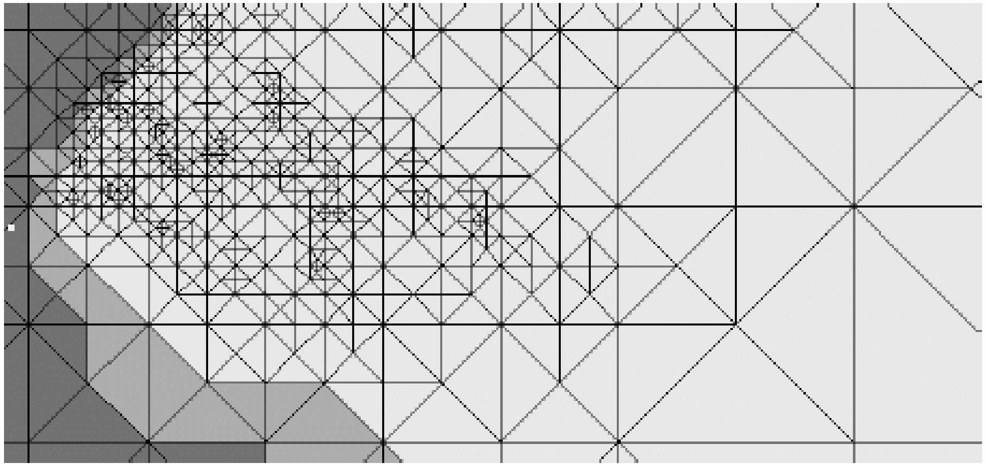
- Same resolution regardless of need
- Same resolution regardless of distance to world

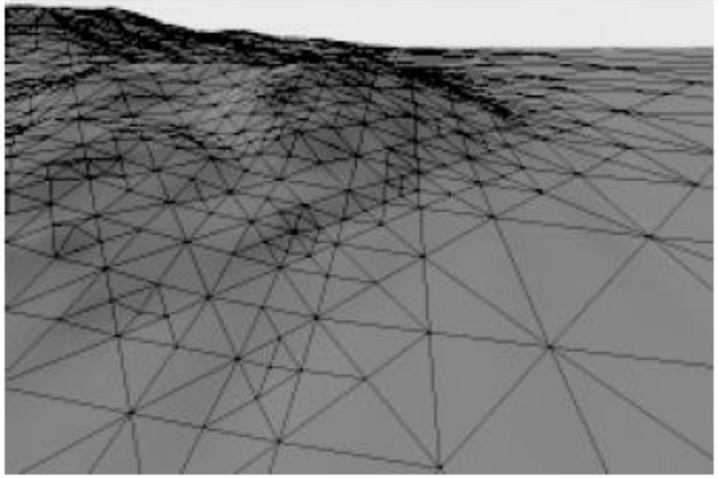
ROAM

Real-time Optimally Adapting Meshes

- Based on Binary Triangle Tree
- Define how to move between higher and lower resolution
- Constantly adapt based on distance



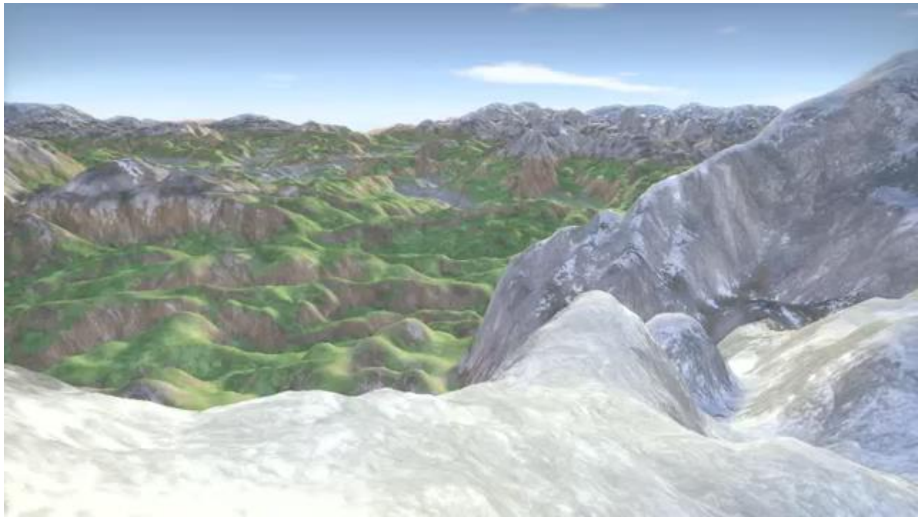


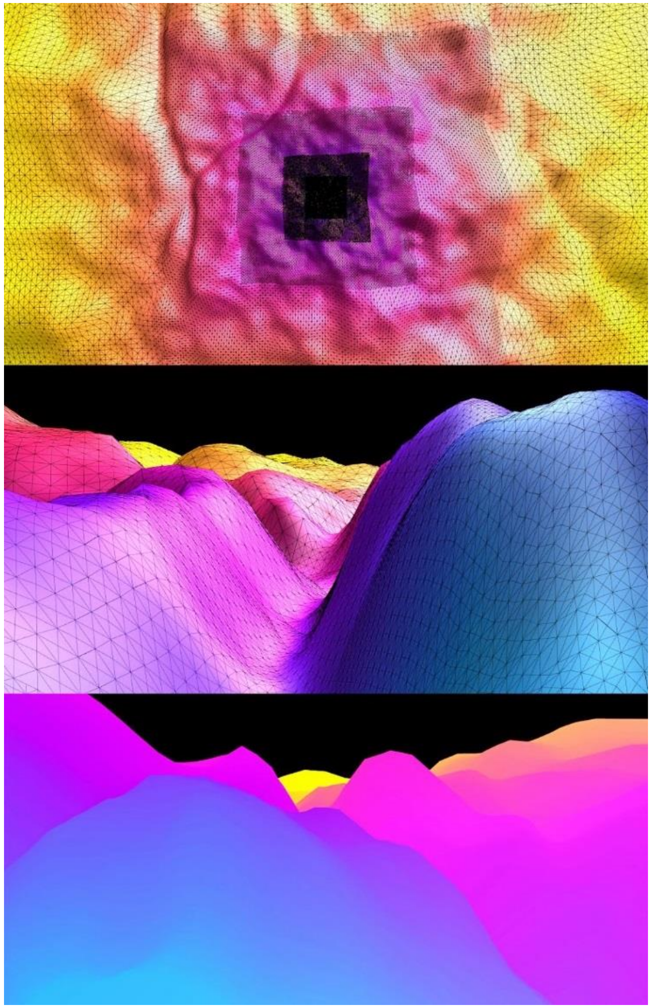


https://graphics.llnl.gov/ROAM/roam.pdf

Today

- Use GPU power
- Submit a mesh
- Middle/close to camera: High resolution
- Fringes: Low Resolution
- Move it to the correct positions in the vertex shader

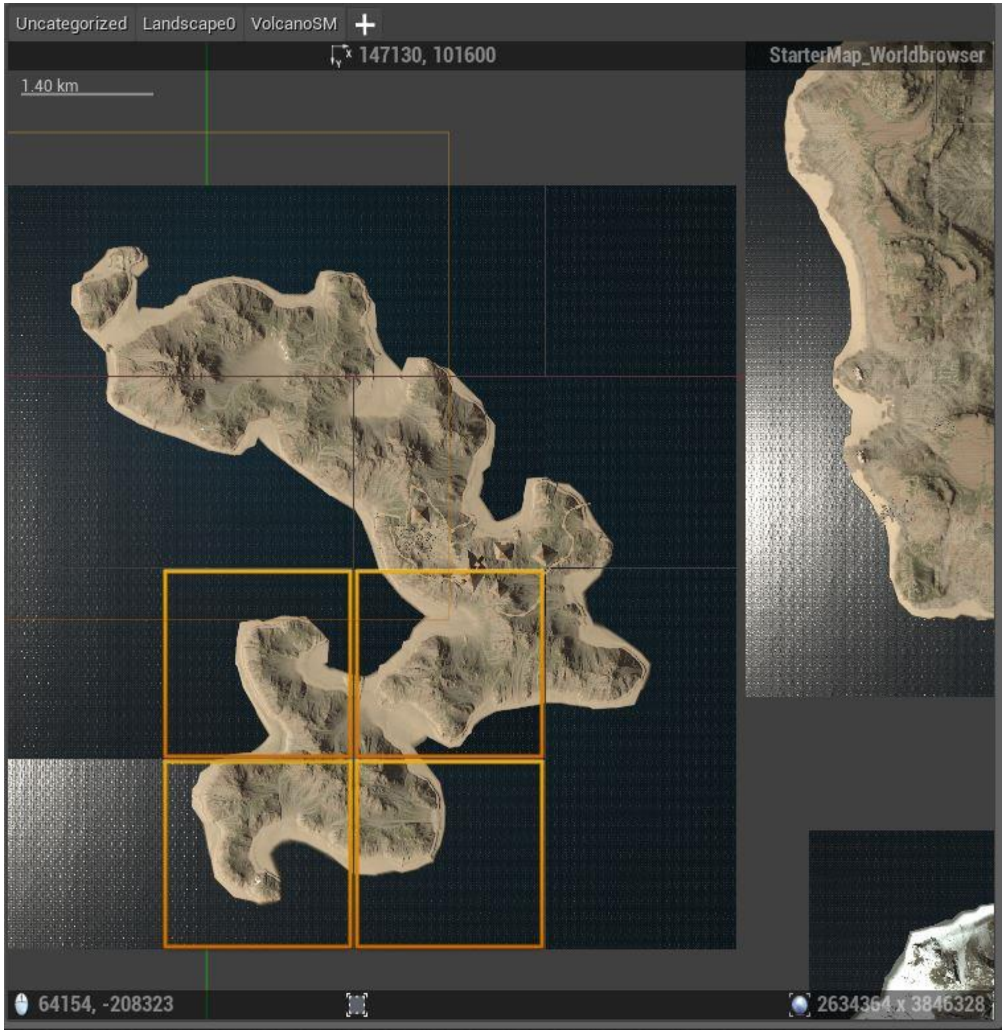




http://casual-effects.blogspot.de/2014/04/fast-terrain-rendering-with-continuous.html

Level Streaming

- **Included in current game engines**
- **Very course geometry streaming**
- **Need to watch out for objects and data**
  - Pathfinding
  - AIs
  - …



Minecraft

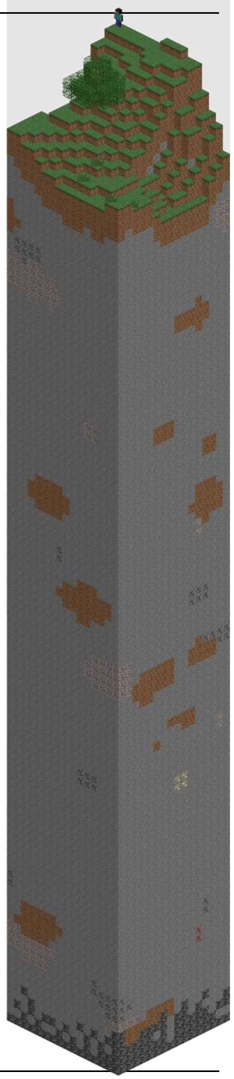
Chunk = 16 x 16 x 256 blocks

Default area of interest (multiplayer): 21x21 chunks around the player

- Inside the area: Normal simulation
- Outside the area: Serialized to disk, nothing updated

When streaming new chunks

- Unknown chunks: PCG
- Known chunks: Load from data



Handling game objects in streaming levels

Naive approach: We only simulate the things around us

E.g. a role-playing game: Only the NPCs around us

- Follow their daily routines
- At day, work, at night, stay at home/in an inn

Player comes into vicinity

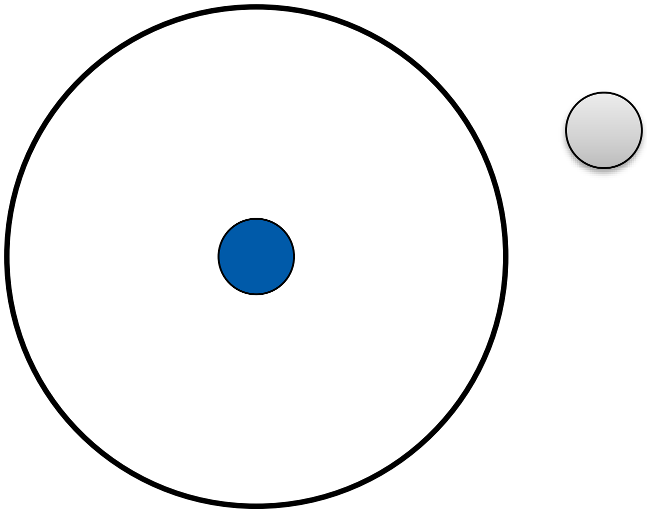
- Spawn as if they were completely new game objects
- Tick them all the time, even if they are not visible
- Forward the simulation to an appropriate state
- Reduced LOD version

Spawn as new objects

Simple to implement

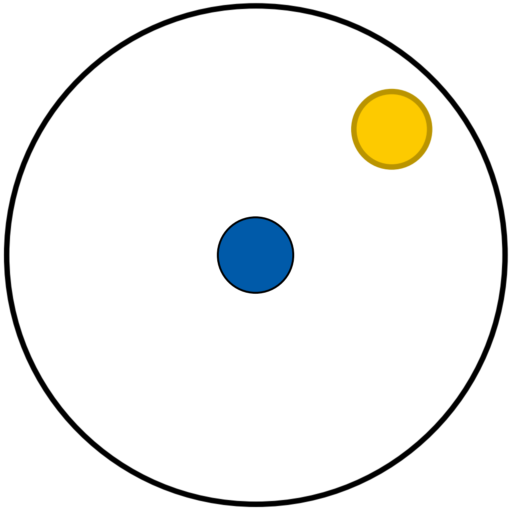
Depending on the game, effects can range from unnoticeable to game-breaking

Especially noticeable if NPCs always follow a script



Spawn as new objects

- Simple to implement
- Depending on the game, effects can range from unnoticeable to game-breaking
- Especially noticeable if NPCs always follow a script



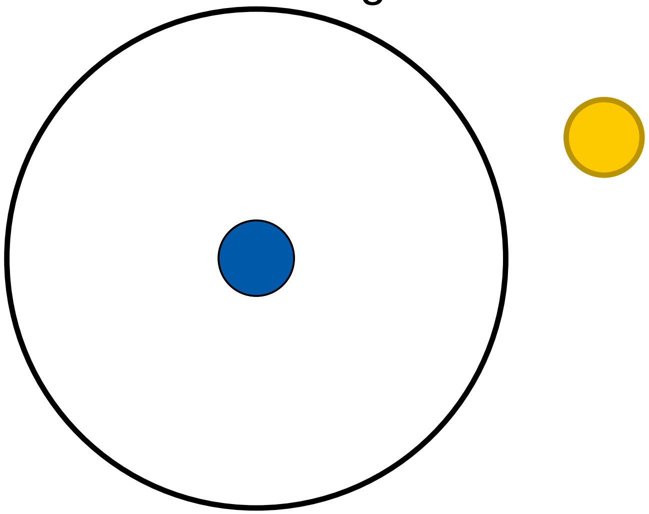
Continuously updating all objects

+The states of the objects are always correct

- Can be very costly

Interdependencies with other objects and the level

- Moving NPCs need to know about the level geometry to navigate
- NPCs who interact with each other
- Can counteract the idea of streaming



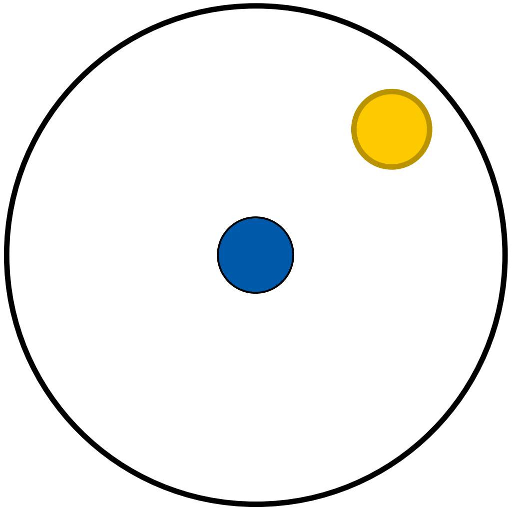
Continuously updating all objects

+The states of the objects are always correct

- Can be very costly

Interdependencies with other objects and the level

- Moving NPCs need to know about the level geometry to navigate
- NPCs who interact with each other
- Can counteract the idea of streaming



Forward to an appropriate state

Similar to recreating persistent objects (e.g. for networked games or save games)

Example

- An NPC in a RPG
- When it left the player's streaming radius the last time, its state was persisted
- After spawning again, the state is restored and forwarded
  - E.g. if 4 hours passed, the NPCs stamina/hunger attribute is reduced by an equivalent amount

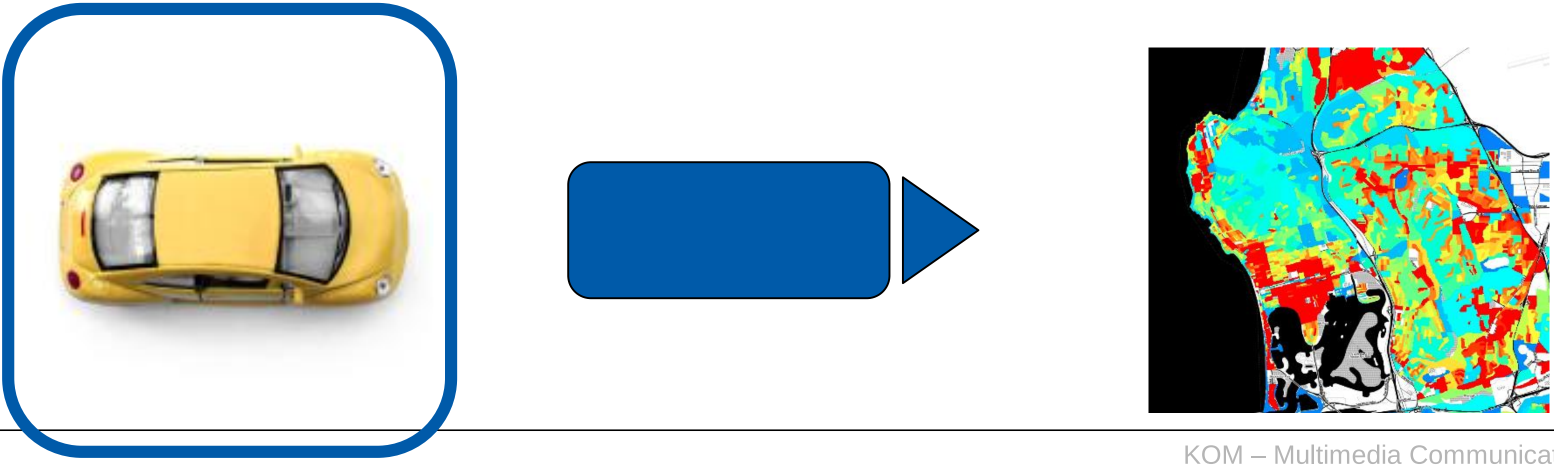
No ticks while not active

Reduced LOD version

When objects go out of relevancy, they get replaced by LOD versions

Example: GTA-style system

- In relevancy radius: Fully accurate version of the car
- Close to relevancy radius: Fully accurate version, so cars don’t pop up once they are close enough
- Further away: Simulated version only, simplified collision and navigation
- Far away: Replaced by a simulation of the traffic density in a whole block



Sound

mp3 and similar compressed formats

- Nothing special – at least not anymore

Coarse streaming for sound effects

- Easy
  - Sound effects are short
  - Sound effects don’t stay on screen
  - Sound effects can stay in CPU RAM

Fine grained streaming for music and maybe speech

- Even mp3 players do it

Large and small scale simultaneously

Really Big Worlds

32 bit floats

- "total precision is 24 bits (equivalent to $\log_{10}(2^{24}) \approx 7.225$ decimal digits)"
  - Can be a little tight for big worlds

Use 64 bit floats for positions

- Hard to integrate 32 bit physics engines

Split and Shift the world

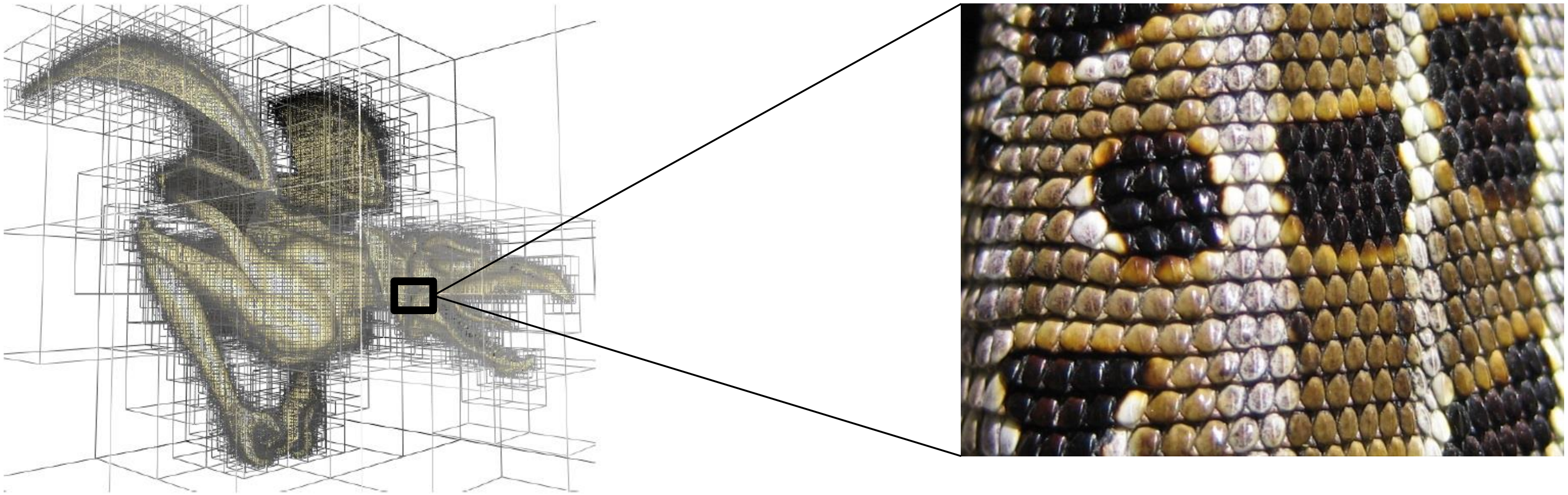
- Split the world
- Shift the closest parts to a position nearer at the camera

Combine PCG and Streaming

As in Minecraft – Generate parts of the world not yet seen

„Infinite Detail“ Engines

- Fill in PCG details after a certain level of detail



Summary

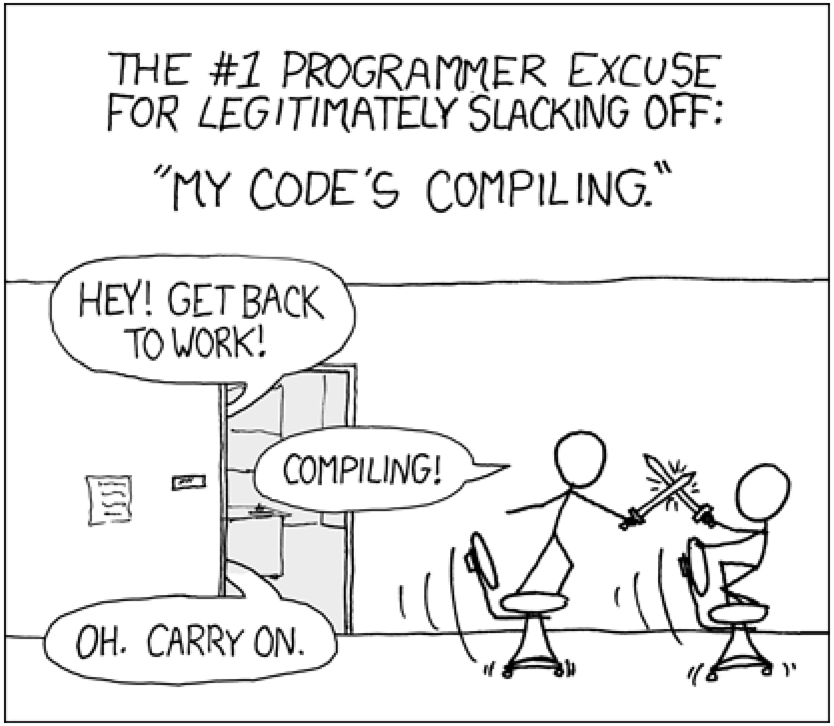
Handling more data than we can fit into memory at once

- Reduce the actual data
- Compress the data
- Load only the data that is needed when it is needed

Don‘t let the player notice

- Low Quality/Quality jumps/Compression artifacts
- LOD switches
- Streamed changes popping up

C++ - Managing large codebases



Translation Units

"A C program need not all be translated at the same time. The text of the program is kept in units called source files, (or preprocessing files) in this International Standard. A source file together with all the headers and source files included via the preprocessing directive #include is known as a preprocessing translation unit. After preprocessing, a preprocessing translation unit is called a translation unit."

→The more headers we include, the larger the translation units get

Forward declarations

Header

```
#include „Foo.h"
Foo* bar;
```

.cpp:

```
#include „Header.h"
bar->doSomething();
```

Header

```
class Foo;
Foo* bar;
```

.cpp

```
#include „Header.h"
#include „Foo.h"

bar->doSomething();
```

Minimizing number of headers

Include only the things needed to compile

Include as much as possible only in the source file

- Prevents chains of includes
- Even if included only once each, will pull unneeded things into the headers

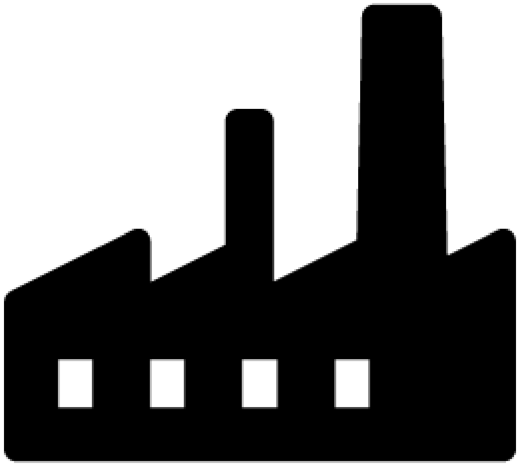
Use forward declarations

- Also helps with cyclical dependencies (if they arise from the design in the first place)

→Clashes with inlining

→Clashes with templates

Leaking need for includes



Hide construction of specific objects in Factory classes

→ Only the factory needs the includes for the specific classes

Precompiled Header

Standard header for everything in your project

Preprocessed to a binary format that is much faster for the compiler

But: Needs to be recreated when something changes

- Even only one header

→ Balance between PCH recreation and compilation speedup

- Easy: Game PCH includes everything in the engine and libraries (should not change)
- Harder: Classes that change little and are often re-used

"Unity builds"

Lump all cpp files together

Used by Unreal

Can load whole file to memory, can lump all includes together

But

- Need to recompile much more on average for a single change
- Include hell if order is changed an includes are not correct

Modules

Split up code into different modules

Ideally, no need to recompile all modules all the time

Can help with better API design

But harder if enforced in the middle of the project

Conclusion

Design upfront

Use Design Patterns

Handle includes well

- As few as possible
- Use forward declarations
- Use PCH appropriately

Balance options

- What goes in PCH
- Unity builds

Thanks

thank you
danke
gracias
merci
obrigado
teşekkür ederim
ngiyabonga
спасибо
dziękuję
bedankt
go raibh maith agat
mochchakkeram
sukriya
kop khun krap
grazie
arigatô
dakujem
terima kasih
감사합니다
ありがとう
мерси