Batching for the masses...
One glCall to draw them all

Samuel Gateau - DevTech Engineer - Nvidia
sgateau@nvidia.com

Once upon a time…

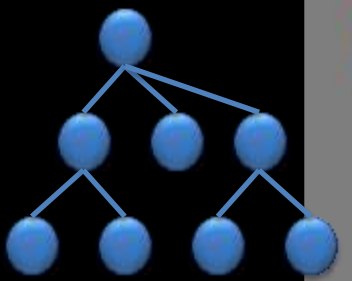
- A large CAD model to render

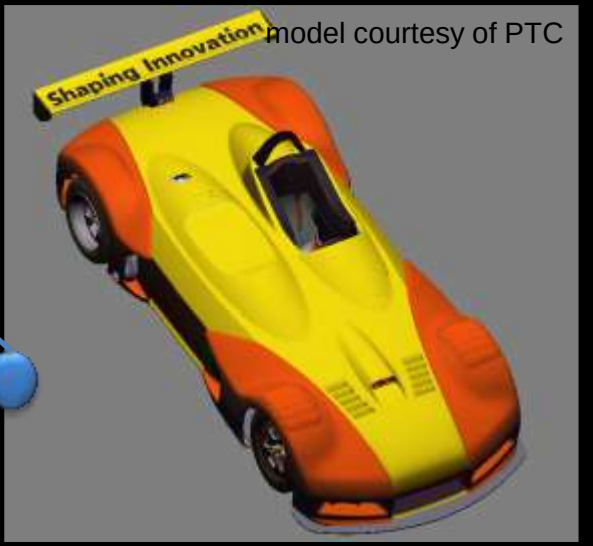
model courtesy of PTC

KHRONOS GROUP

Once upon a time…

- A large CAD model to render
- Described by a scene graph

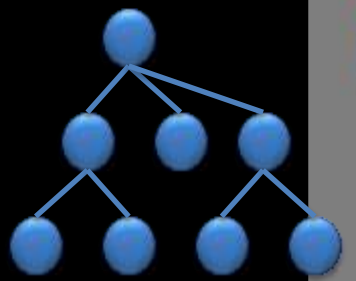


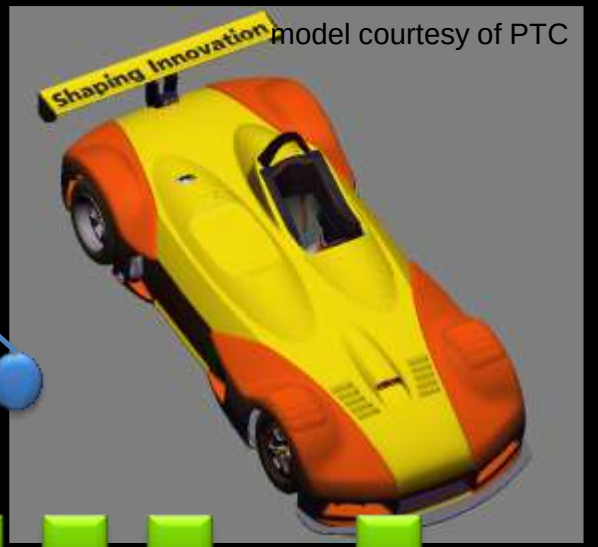


model courtesy of PTC

Once upon a time…

- A large CAD model to render
- Described by a scene graph
- Flatten into a list of shapes to render

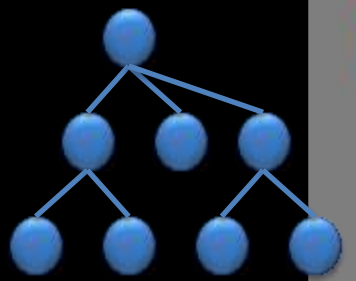


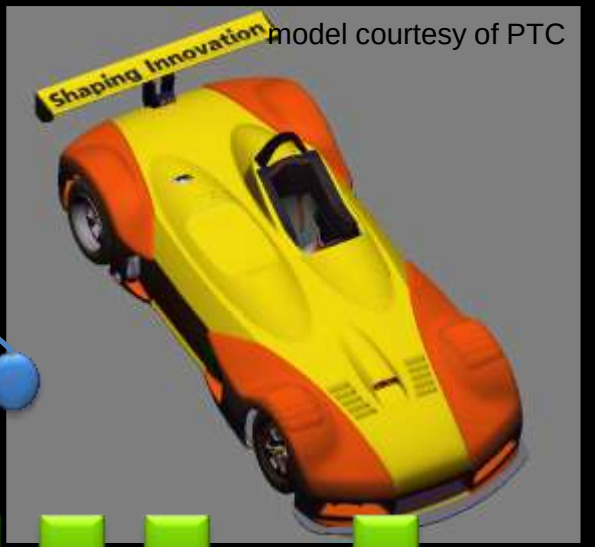


model courtesy of PTC

Once upon a time…

- A large CAD model to render
- Described by a scene graph
- Flatten into a list of shapes to render
- For each shape
  - Geometry
    - Vertex/Index BO
    - Divided into parts (CAD features)

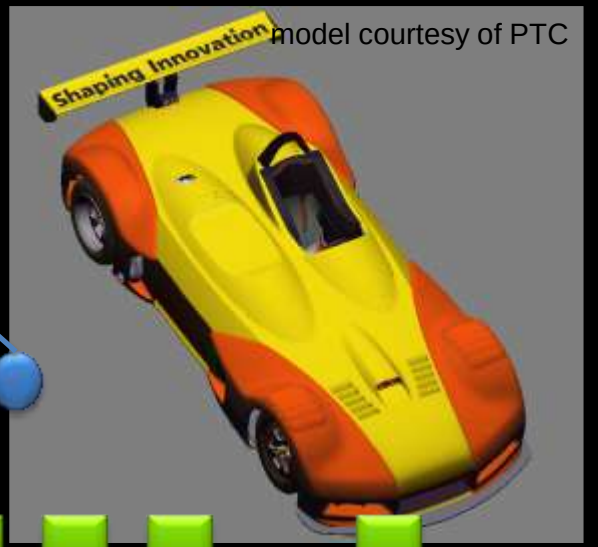




model courtesy of PTC

Once upon a time…

- A large CAD model to render
- Described by a scene graph
- Flatten into a list of shapes to render
- For each shape
  - Geometry
    - Vertex/Index BO
    - Divided into parts (CAD features)
  - Materials (referenced per part)

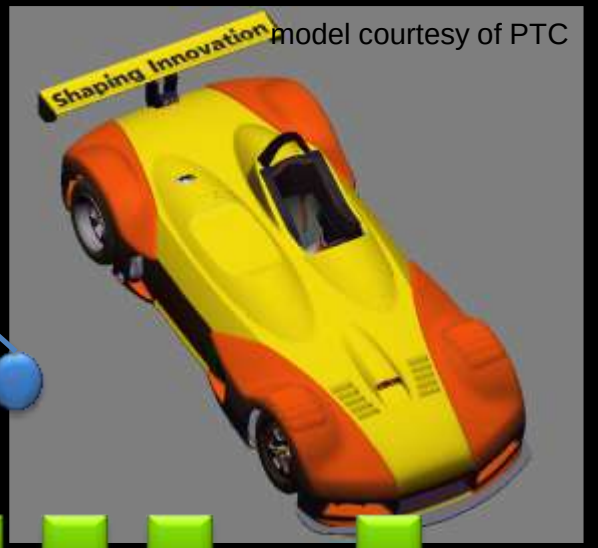


model courtesy of PTC

KHRONOS GROUP

Once upon a time…

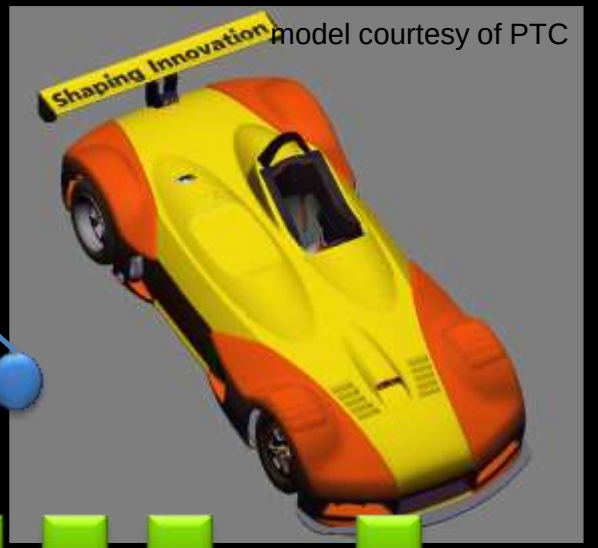
- A large CAD model to render
- Described by a scene graph
- Flatten into a list of shapes to render
- For each shape
  - Geometry
    - Vertex/Index BO
    - Divided into parts (CAD features)
  - Materials (referenced per part)
  - Transform Stack



model courtesy of PTC

Once upon a time…

- A large CAD model to render
- Described by a scene graph
- Flatten into a list of shapes to render
- For each shape
  - Geometry
    - Vertex/Index BO
    - Divided into parts (CAD features)
  - Materials (referenced per part)
  - Transform Stack
- Many drawcalls…



model courtesy of PTC

KHRONOS GROUP

Performance baseline

- Kepler Quadro K5000, i7
- vbo bind and drawcall per part,

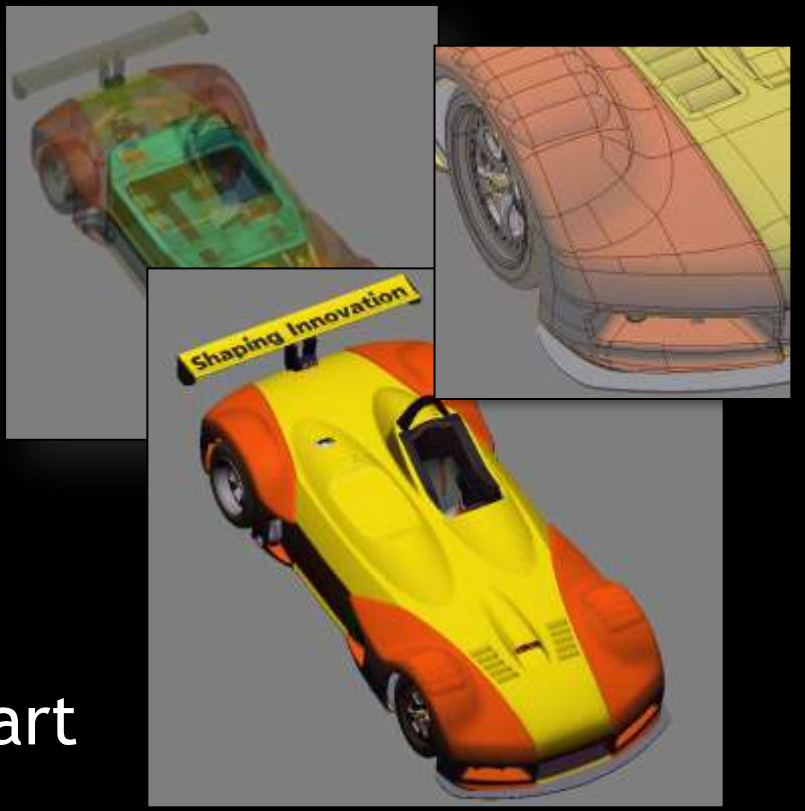
  i.e. 99 000 drawcalls

  scene draw time > 38 ms (CPU bound)
- vbo bind per geometry, drawcalls per part

  scene draw time > 14 ms (CPU bound)
- Batching for fast material/matrix switching

  scene draw time < 3 ms

  1.8 ms with occlusion culling



model courtesy of PTC

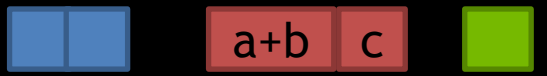
- 99000 total parts, 3.8 Mtris, 5.1 Mverts
- 700 geometries (contain parts), 128 materials
- 2000 objects (reference geometries, materials)

Batching within Object Geometry

Group and combine parts with same state

- Object‘s list must be rebuilt based on material/enabled state
- Use ARB_multi_draw_indirect for single drawcall!

Parts with different materials in geometry



Grouped and „grown“ drawcalls

Single drawcall with material/matrix changes

```
DrawElementsIndirect
{
   GLuint       count;
   GLuint       instanceCount; ← 1
   GLuint       firstIndex;
   GLint        baseVertex;
   GLuint       baseInstance;  ← encode matrix/material index
}

DrawElementsIndirect  objectsDrawCalls[ N ];
```

OpenGL 2 approach

- Parameters:
  - Avoid many small uniforms
  - Arrays of uniforms, grouped by frequency of update, tightly-packed

```
uniform mat4 worldMatrices[2];
uniform vec4 materialData[8];

#define matrix_world   worldMatrices[0]
#define matrix_worldIT worldMatrices[1]

#define material_diffuse  materialData[0]
#define material_emissive materialData[1]
#define material_gloss    materialData[2].x

// GL3 can use floatBitsToInt and friends
// for free reinterpret casts within
// macros
...

  wPos = matrix_world * oPos;
...
// in fragment shader
  color = material_diffuse +
          material_emissive;
...
```

OpenGL 2 approach

```
foreach (obj in scene) {
  if ( isVisible(obj) ) {

    setupDrawGeometryVertexBuffer (obj);

    glUniformMatrix4fv (... matrices[ obj.matrix ] );

    // iterate over different materials used
    foreach ( batch in obj.materialCaches) {

      glUniform4fv (... materials[ batch.material ] );

      glMultiDrawElements (GL_TRIANGLES, batch.counts, GL_UNSIGNED_INT ,
                           batch.offsets,batched.numUsed);
    }
  }
}
```

OpenGL 4.4 ARB approach

- Parameters:
  - TextureBufferObject (TBO) for matrices
  - UniformBufferObject (UBO) with array data to save costly binds
  - NEW ARB_shader_draw_parameters extension for gl_BaseInstanceARB access
  - Caveat: costs for indexed fetch for parameters (balance CPU/GPU boundedness)

```
uniform  samplerBuffer matrixBuffer;
```

```
uniform  materialBuffer {
  Material materials[128];
};
```

```
// encoded assignments in 32-bit
ivec2 vAssigns =
   ivec2(gl_BaseInstanceARB >> 16,
         gl_BaseInstanceARB & 0xFFFF);
flat out ivec2 fAssigns;

// in vertex shader
  fAssigns = vAssigns;

  worldTM = getMatrix (matrixBuffer,
                       vAssigns.x);
...
// in fragment shader
  color = materials[fAssigns.y].diffuse...
```

OpenGL 4.4 ARB approach

- Parameters:
  - NEW NV/ARB_bindless_texture to store texture handles as 64bit values inside buffers
  - `uint64 glGetTextureHandle ( tex )`
  - `glMakeTextureHandleResident ( hdl )`

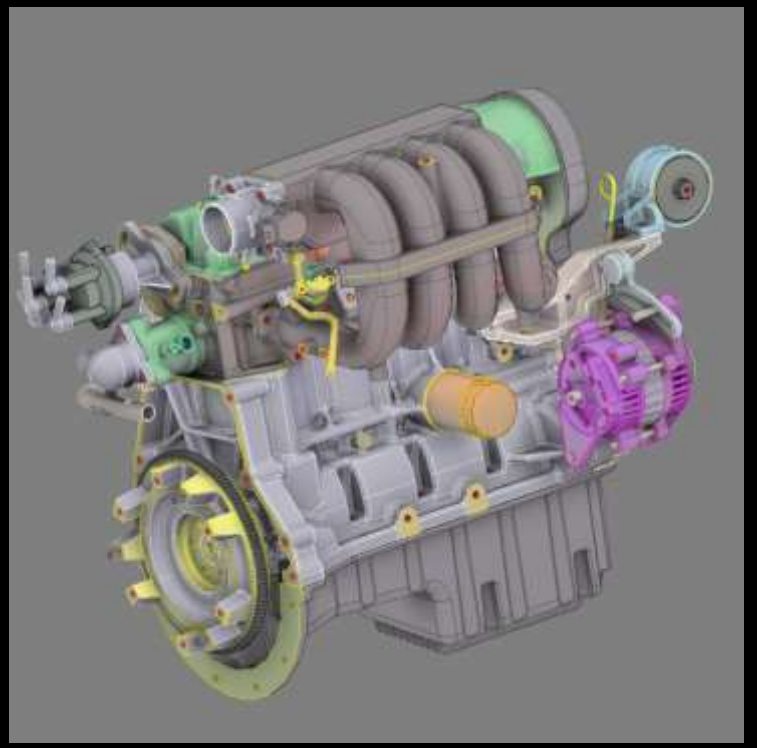
```
// NEW ARB_bindless_texture stored inside buffer!

struct Material {
  sampler2D albedo; // now allowed :)
  vec4 albedo_weight;
  ...
};

uniform  materialBuffer {
  Material materials[128];
};

// in fragment shader

flat in ivec2 fAssigns;
...
color = texture (materials[fAssigns.y].albedo...
```

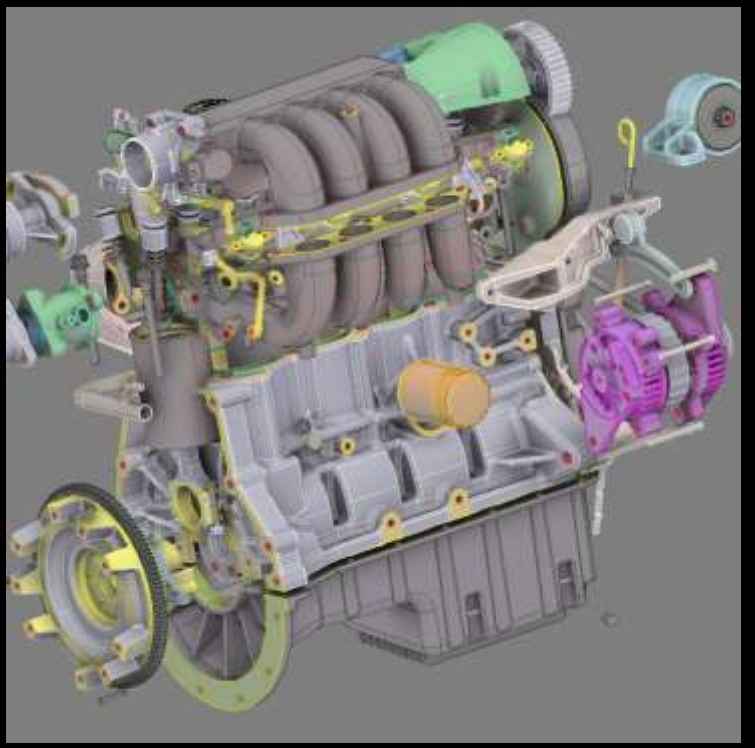
OpenGL 4.4 ARB approach

```
setupSceneMatrixAndMaterialBuffer (scene);

foreach ( obj in scene.objects ) {
  ...

  // we encode the assignments in baseInstance, nothing needed ☺
  // ARB_shader_draw_parameters

  // draw everything in one go
  glMultiDrawElementsIndirect ( GL_TRIANGLES, GL_UNSIGNED_INT,
                                obj->indirectOffset, obj->numIndirects, 0 );
}
```

Transform Tree Updates

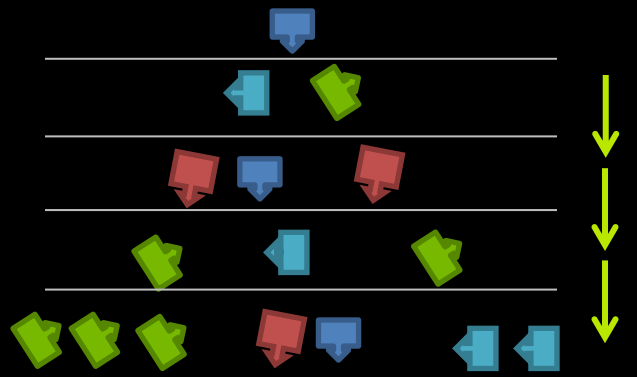
- How about animation? Now that all matrices live on GPU

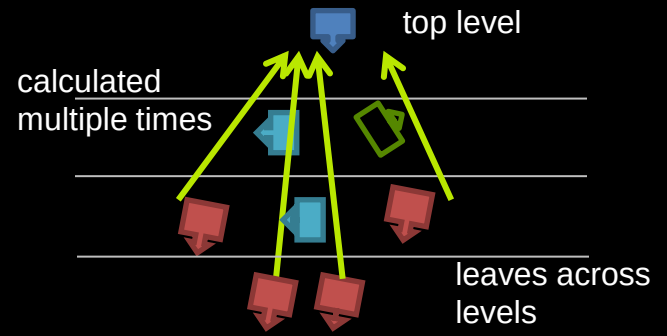




Transform Tree Updates

- Upload only matrices that changed, use ARB_compute_shader for tree update
- Level-wise procesing could be too little work, must wait on previous level
- Leaf-wise processing means more work generated, despite calculating some matrices multiple times
  - Use temp matrix, local path on stack to top level node, only stores, no reads from „shared“ nodes
- Mix both depending on tree

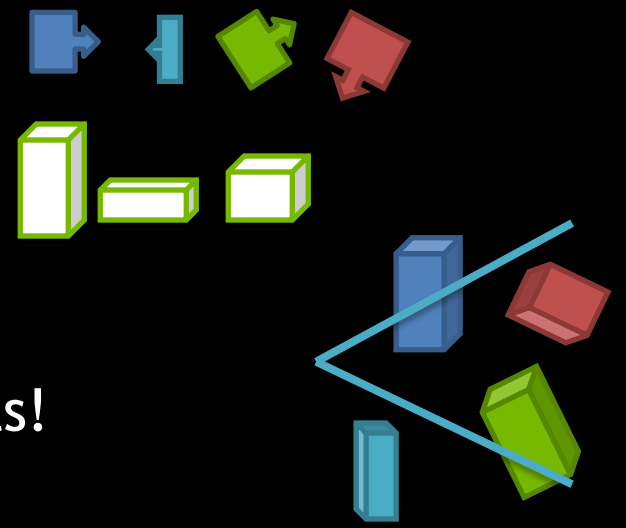




One thread per leaf, generate work queue by leaf-level, to avoid divergence

GPU Culling Basics

- GPU friendly processing
  - Matrix and bbox buffer, object buffer
  - XFB/Compute or „invisible“ rendering
  - Vs. old techniques: Single GPU job for ALL objects!
- Results
  - „Readback“ GPU to Host
    - Can use GPU to pack into bit stream
  - „Indirect“ GPU to GPU
    - Set DrawIndirect‘s instanceCount to 0 or 1
    - Or use ARB_indirect_parameters



```
0,1,0,1,1,1,0,0,0

buffer cmdBuffer{
  Command cmds[];
};
...
cmds[obj].instanceCount = visible;
```

Recap

- Parameter and Geometry batching
  - MultiDrawIndirect and ARB_shader_draw_parameters as well as family of bindless extensions save CPU costs for „low complexity“ objects
  - ARB_compute_shader to compute instead of transfer
- Readback vs Indirect Culling
  - Readback variant „easier“ to be faster (no setups...), but syncs!
  - (NV_bindless)/ARB_multidraw_indirect/ARB_indirect_parameters as mechanism for GPU creating its own work, research and feature work in progress

Thanks,

- Special thanks to Christoph Kubrisch who lead this research.
- For more details
  - **GTC : S3032 - Advanced Scenegraph Rendering Pipeline**
  - http://on-demand.gputechconf.com/gtc/2013/presentations/S3032-Advanced-Scenegraph-Rendering-Pipeline.pdf

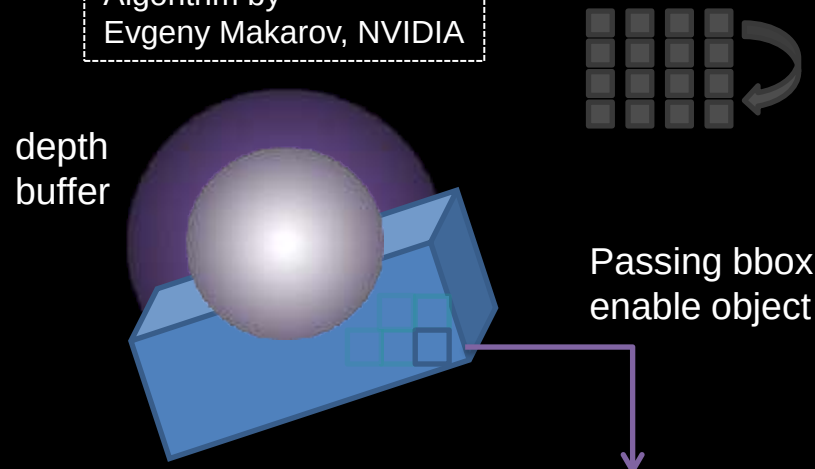
questions ?

sgateau@nvidia.com

KHRONOS GROUP

Occlusion Culling

Algorithm by
Evgeny Makarov, NVIDIA



- OpenGL 4.2+
  - Depth-Pass
  - Raster „invisible“ bounding boxes
    - Disable Color/Depth writes
    - Geometry Shader to create the three visible box sides
    - Depth buffer discards occluded fragments (earlyZ...)
    - Fragment Shader writes output: visible[objindex] = 1

THROW OUT ?

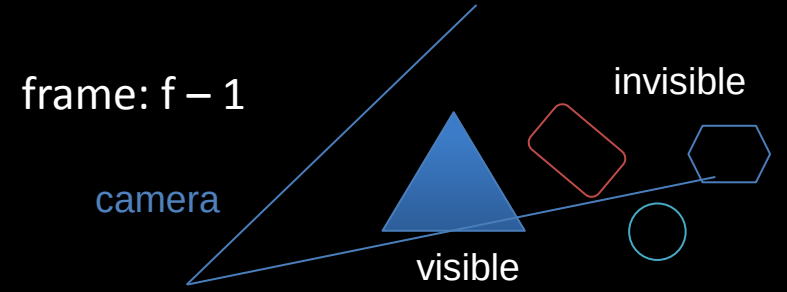
```
// GLSL fragment shader
// from ARB_shader_image_load_store
layout(early_fragment_tests) in;

buffer visibilityBuffer{
  int visibility[];
};

flat in int objID;
void main(){
 visibility[objID] = 1;
}
// buffer would have been cleared
// to 0 before
```

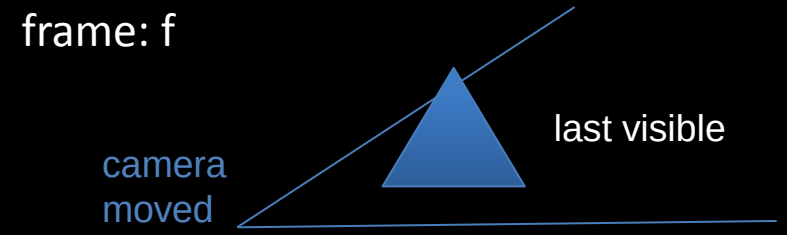
KHRONOS GROUP

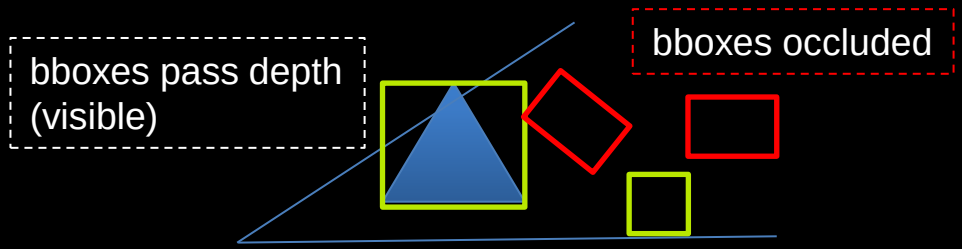
Temporal Coherence

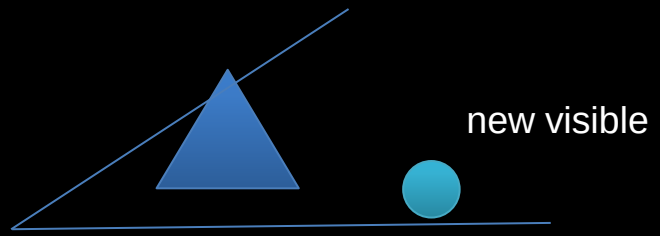
- Exploit that majority of objects don‘t change much relative to camera
- Draw each object only once (vertex/drawcall-bound)
  - Render last visible, fully shaded
    `(last)`
  - Test all against current depth:
    `(visible)`
  - Render newly added visible:
    none, if no spatial changes made
    `(~last) & (visible)`
  - `(last) = (visible)`

THROW OUT ?

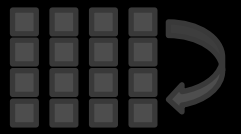






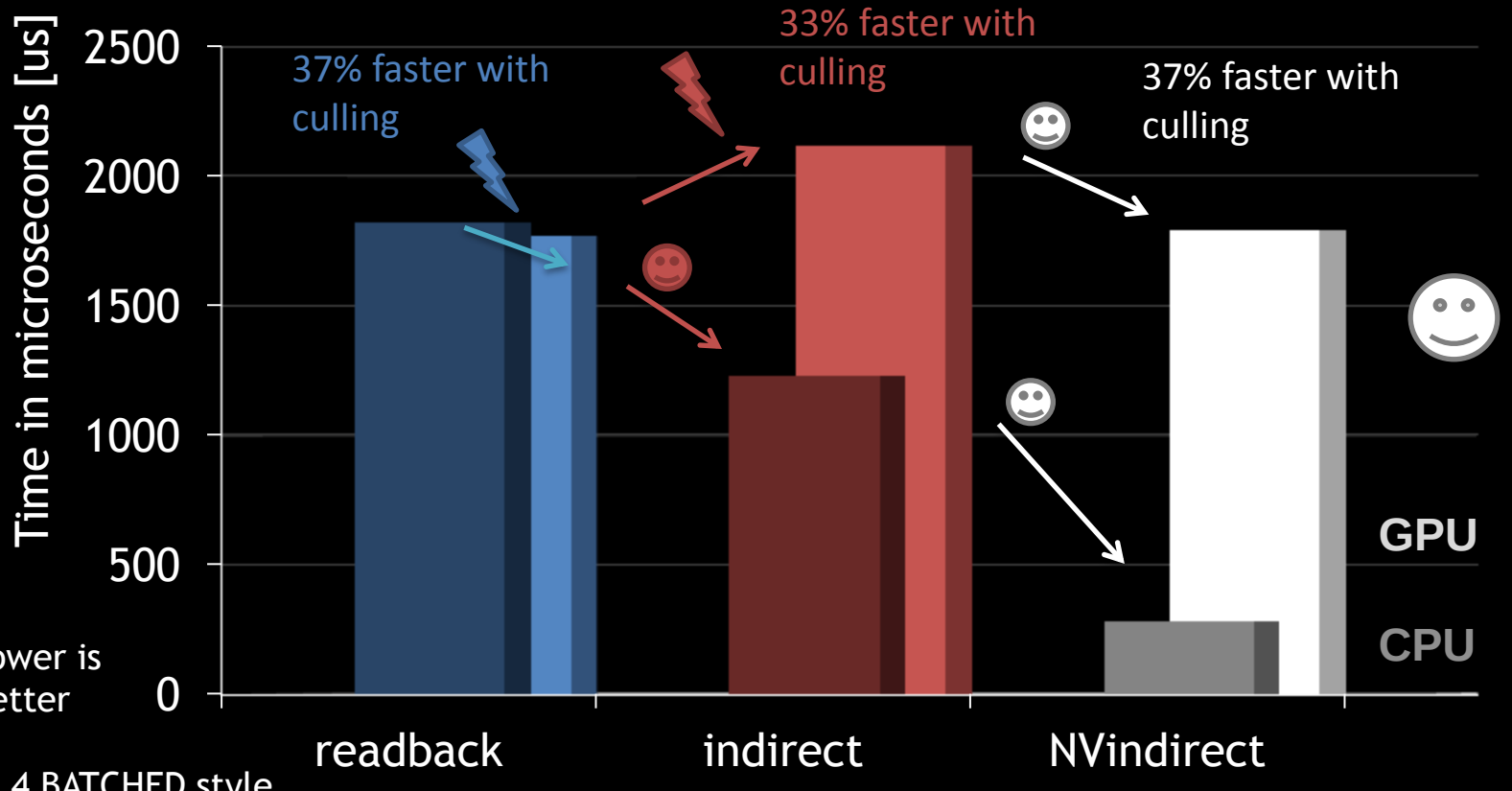


Culling Readback vs Indirect



For readback results, CPU has to wait for GPU idle

In the „draw new visible“ phase indirect cannot benefit of „nothing to setup/draw“ in advance, still processes „empty“ lists



GL4 BATCHED style

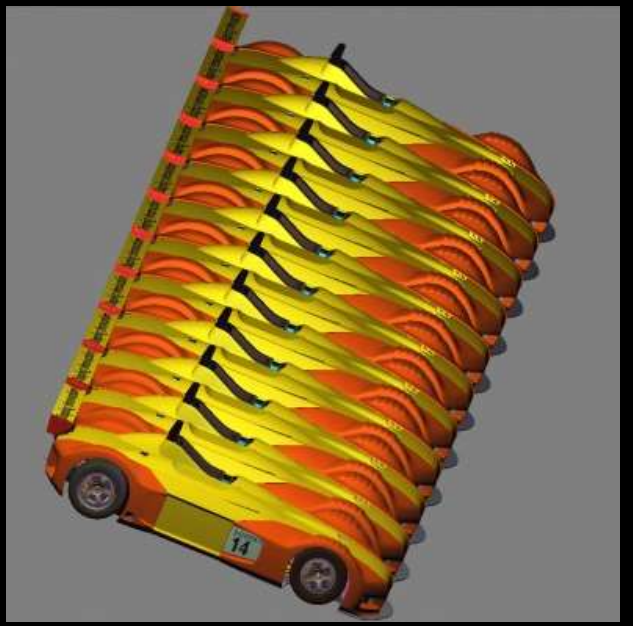
THROW OUT ?

NV_bindless_ multidraw_indirect saves CPU and bit of GPU time

Your milage may vary, depends on triangles per drawcall and # of „invisible“

Will it scale?

- 11 x the car
  - everything but materials duplicated in memory, NO instancing
  - 1m parts, 36k objects, 40m tris, 37m verts



THROW OUT ?

- stall-free culling 78 fps
  - Only 1 ms CPU time using NV_bindless_multidraw_indirect and occlusion culling as described
  - Work in progress, should reach > 100 fps

- Thank you!
  - Contact
    - ckubisch@nvidia.com

KHRONOS GROUP