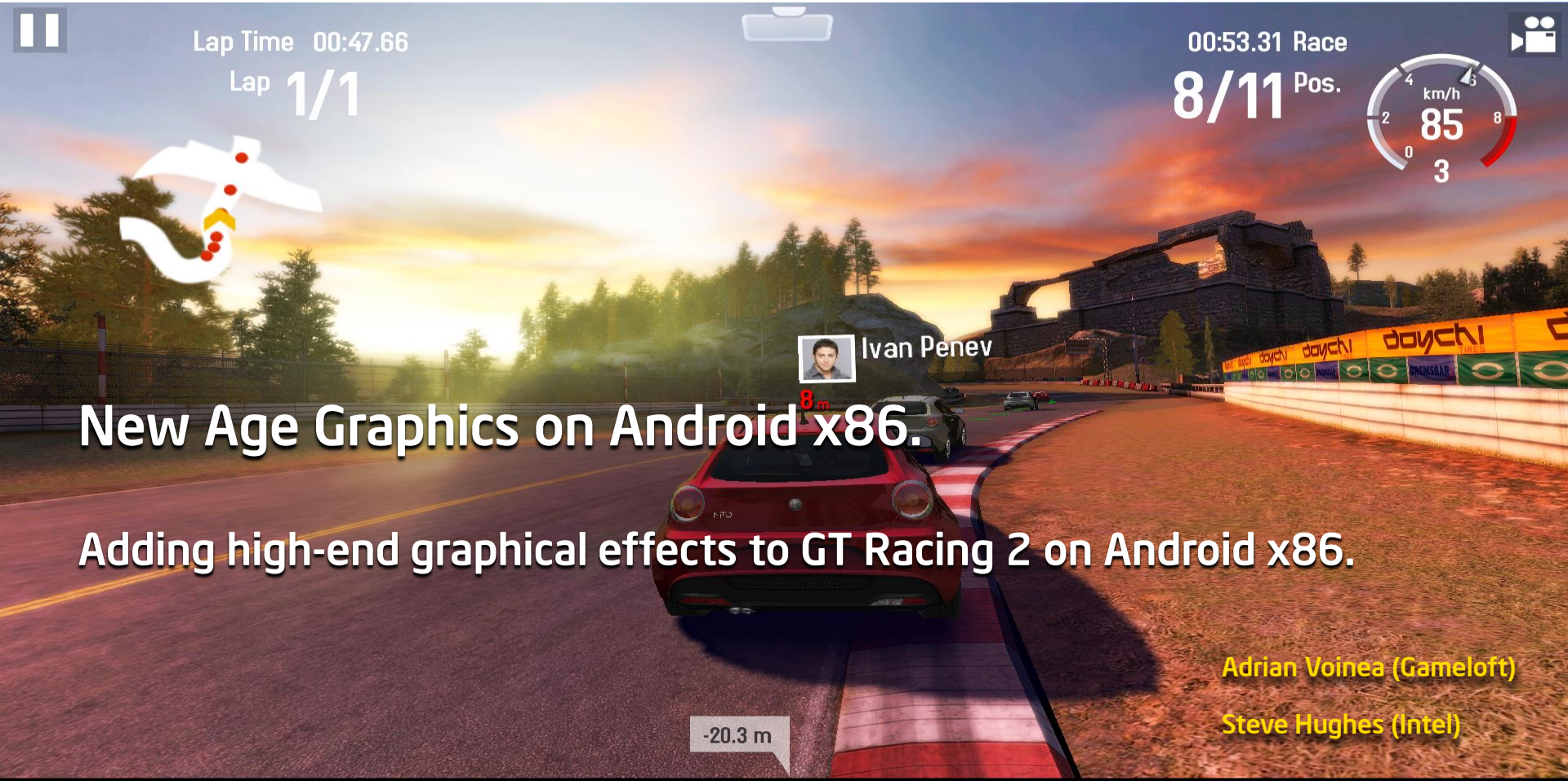


New Age Graphics on Android x86.

Adding high-end graphical effects to GT Racing 2 on Android x86.

Adrian Voinea (Gameloft)

Steve Hughes (Intel)

Legal

*Other names and brands may be claimed as the property of others.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED,
BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS
OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice.

All products, dates, and figures specified are preliminary based on current expectations, and are subject to change without notice.

Intel processors, chipsets, and desktop boards may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Any code names featured are used internally within Intel to identify products that are in development and not yet publicly announced for release. Customers, licensees and other third parties are not authorized by Intel to use code names in advertising, promotion or marketing of any product or services and any such use of Intel's internal code names is at the sole risk of the user.

Intel product plans in this presentation do not constitute Intel plan of record product roadmaps. Please contact your Intel representative to obtain Intel’s current plan of record product roadmaps.

Performance claims: Software and workloads used in performance tests may have been optimized for performance only on Intel® microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more information go to
http://www.Intel.com/performance

Iris™ graphics is available on select systems. Consult your system manufacturer.

Intel, Intel Inside, the Intel logo, Intel Core and Iris are trademarks of Intel Corporation in the United States and other countries.

GT Racing 2 Intel Introduction

- Gameloft, a leading global publisher with key franchises like Asphalt, Despicable Me, Ice Age Village, Modern Combat decided to join forces with Intel to bring one optimized version of its latest simulation racing game: GT Racing 2
- Our goal was to push their latest hardware, Baytrail for Android to the maximum of its capacities and provide consumers with one of the best playable performance.

- Working on unreleased platform is quite a challenge, but exactly in line with what we do at Gameloft

- In the end, with Intel's support, we manage to deliver a top quality version of this racing title, which you can find only on x86.

GT Racing 2 Intel Introduction

- This is the end result

Original Version

GTR2 Intel Enhanced

GPA Screenshot

GT Racing 2 Intel
Special Effects - Depth of Field

- Active in Main Menu
- Puts emphasis on the car displayed by blurring further objects
- Two blur sub passes, vertical and horizontal, that are merged together in the final composition step

GT Racing 2 Intel
Special Effects - Depth of Field

- Horizontal blur applied to the initial framebuffer
- Output is ¼ of native resolution

GPA Screenshot

GT Racing 2 Intel
Special Effects - Depth of Field

- Vertical blur is applied to the output buffer from the horizontal blur step

GPA Screenshot

GT Racing 2 Intel
Special Effects - Depth of Field

The Depth of Field shader uses a depth difference to control the blur

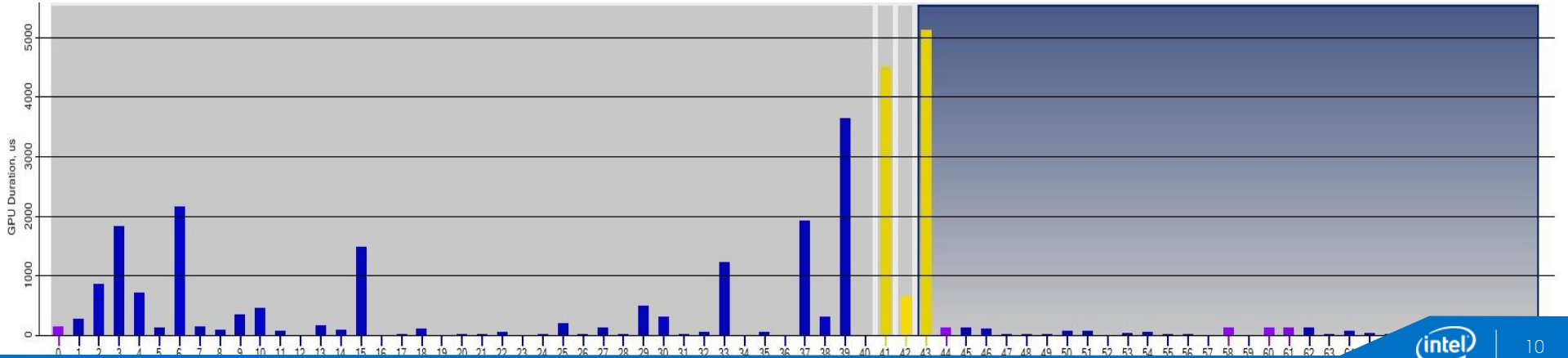
- lowp vec3 color = texture2D(texture0, vCoord0).rgb; ***//unaltered render target***
- lowp vec3 blur = texture2D(texture3, vCoord0).rgb; ***//blurred render target***
- lowp float depthDiff = abs(depth - focusDepth); ***//calculate the depth difference between a chosen focus point***
- depthDiff += smoothstep(0.24, 1.0, length(focusPoint - vCoord0)); ***//take in consideration only the depth value greater then 0.24***
- lowp vec3 dofColor = mix(color, blur, depthDiff); ***//color * (1 - depthDiff) + blur * depthDiff***

GT Racing 2 Intel
Special Effects - Depth of Field

GT Racing 2 Intel
Special Effects - Depth of Field

- Horizontal blur pass: 4.5ms
- Vertical blur pass: 0.66ms
- Final compose: 5.1ms
- **Total:** 10.26 ms to apply for DoF algorithm

GPA Screenshot



GT Racing 2 Intel
Special Effects – Heat Haze

- Heat haze distortion on the start of every race
- Gives the effect of hot air rising from the track

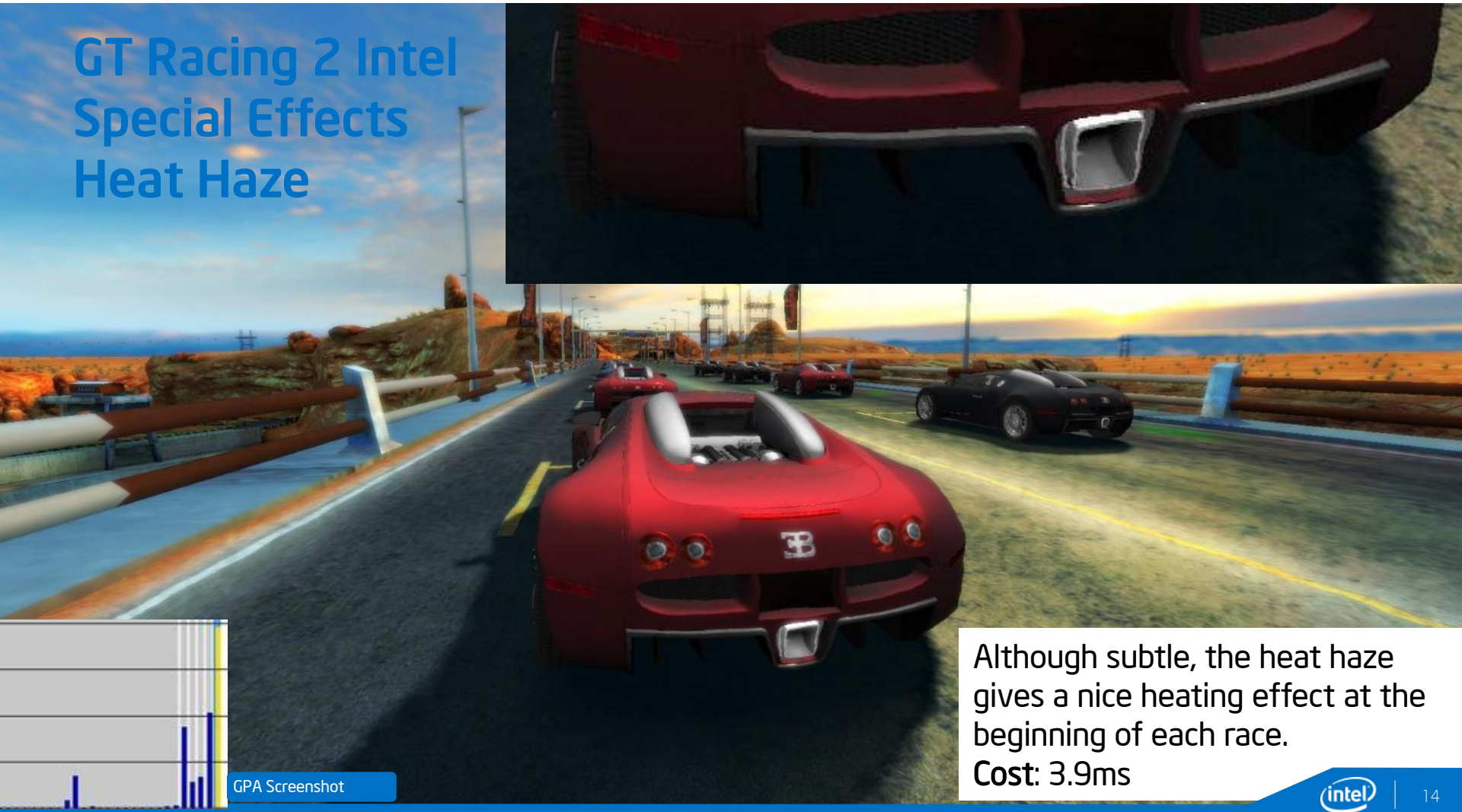
GT Racing 2 Intel Special Effects – Heat Haze

- Starting from the car coordinates, an alpha mask is generated.

GT Racing 2 Intel Special Effects – Heat Haze

- A distortion texture is applied over the mask obtained

GT Racing 2 Intel Special Effects Heat Haze



GPA Screenshot

Although subtle, the heat haze gives a nice heating effect at the beginning of each race.

Cost: 3.9ms

GT Racing 2 Intel
Special Effects - Lightshafts

- Improves game immersion in sunny environments
- It requires several post-processing passes, and the effect can be quite expensive

GT Racing 2 Intel
Special Effects – Lightshafts

- The base render target which will contain the sun will be occluded by the scene objects.

- We need to render just to sun, so we separate blending equations for transparent objects:
  - Solids output 0 to the alpha channel
  - Transparent use separate blending equations:
    - The color is preserved
    - The alpha information is not affecting the desired result from our render target

GT Racing 2 Intel Special Effects – Lightshafts

Radial blur pass

- Applying radial blur starting from the sun position
- The effect requires three passes to smooth out the rays
- This is achieved efficiently for mobiles, by keeping a small sized RTT and using the same shader pair
- All three passes take ~4.4ms

GT Racing 2 Intel
Special Effects – Lightshafts

Radial blur pass

- In the vertex shader , we are computing texture coordinates for the radial blur

  - mediump vec2 center = vec2(center_x, center_y); ***//sun position in uv coordinates***
  - mediump vec2 dir = (center - vCoord0) * scale; ***//radial blur direction***
  - mediump vec2 SampleUVDelta = (dir * blurScale) / 8.0; ***//offset for radial blur***
  - mediump float blurOffset = 0.01;

  - vCoord0 = vCoord0 + (dir * blurOffset);
  - vCoord1 = vCoord0 + SampleUVDelta;
  - vCoord2 = vCoord1 + SampleUVDelta;
  - vCoord3 = vCoord2 + SampleUVDelta;
  - vCoord4 = vCoord3 + SampleUVDelta;
  - vCoord5 = vCoord4 + SampleUVDelta;
  - vCoord6 = vCoord5 + SampleUVDelta;
  - vCoord7 = vCoord6 + SampleUVDelta;

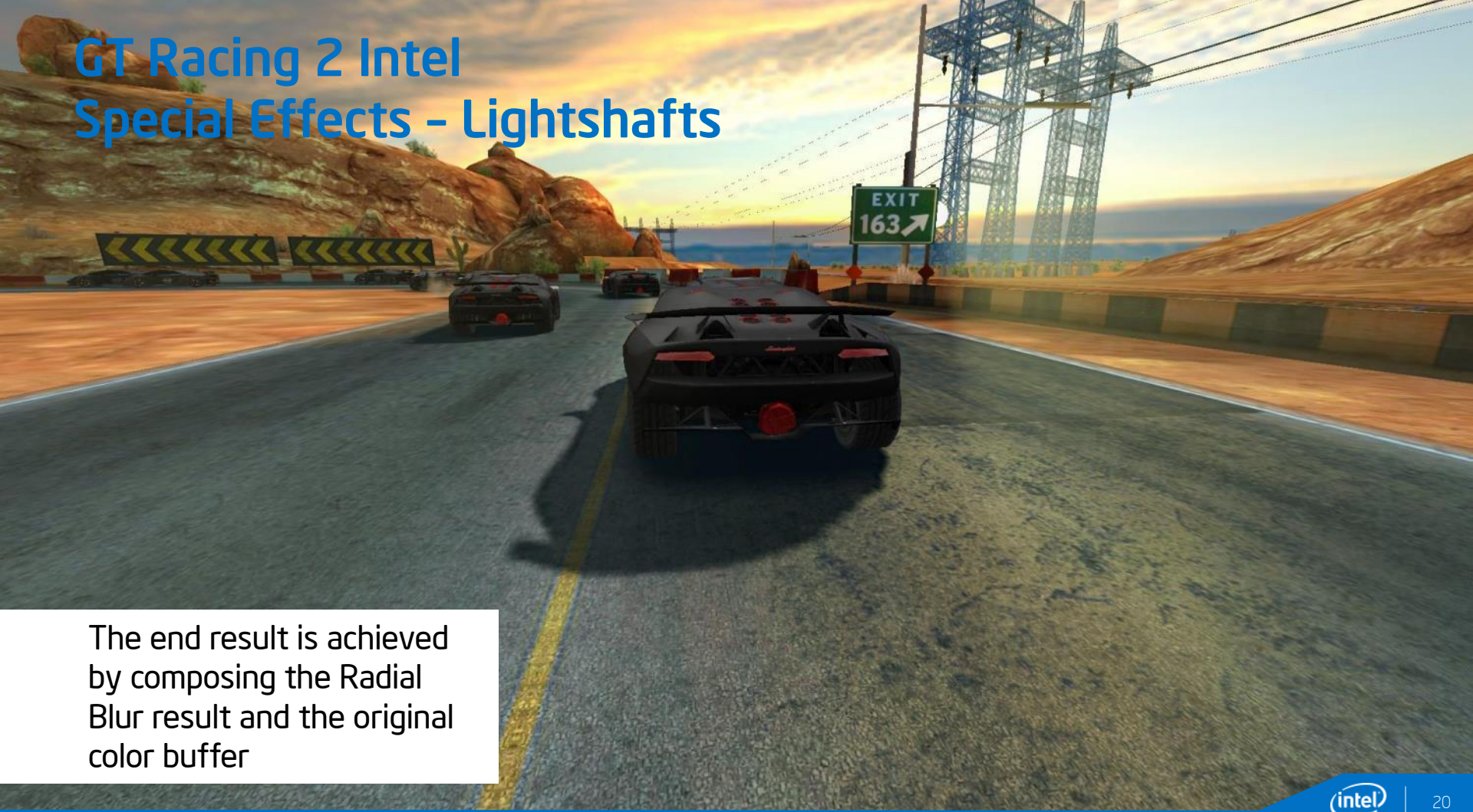
GT Racing 2 Intel
Special Effects – Lightshafts

Radial blur pass

- Inside the fragment shader, we are using the previously computed coordinates to apply radial blur algorithm

  - color += texture2D(texture0, vCoord1).rgb;
  - color += texture2D(texture0, vCoord2).rgb;
  - color += texture2D(texture0, vCoord3).rgb;
  - color += texture2D(texture0, vCoord4).rgb;
  - color += texture2D(texture0, vCoord5).rgb;
  - color += texture2D(texture0, vCoord6).rgb;
  - color += texture2D(texture0, vCoord7).rgb;
  - gl_FragColor.rgb = color / 8.0;

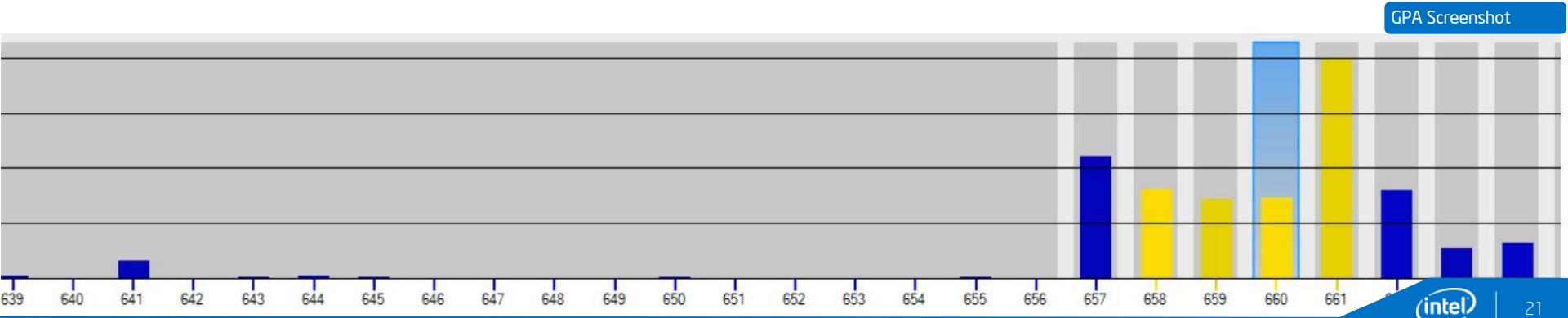
GT Racing 2 Intel Special Effects – Lightshafts



The end result is achieved by composing the Radial Blur result and the original color buffer

GT Racing 2 Intel
Special Effects – Lightshafts

- First pass: 1.6 ms
- Second pass: 1.4 ms
- Third pass: 1.4 ms
- Compose: 4 ms
- **Total**: 8.4 ms



GT Racing 2 Intel Special Effects – Bloom

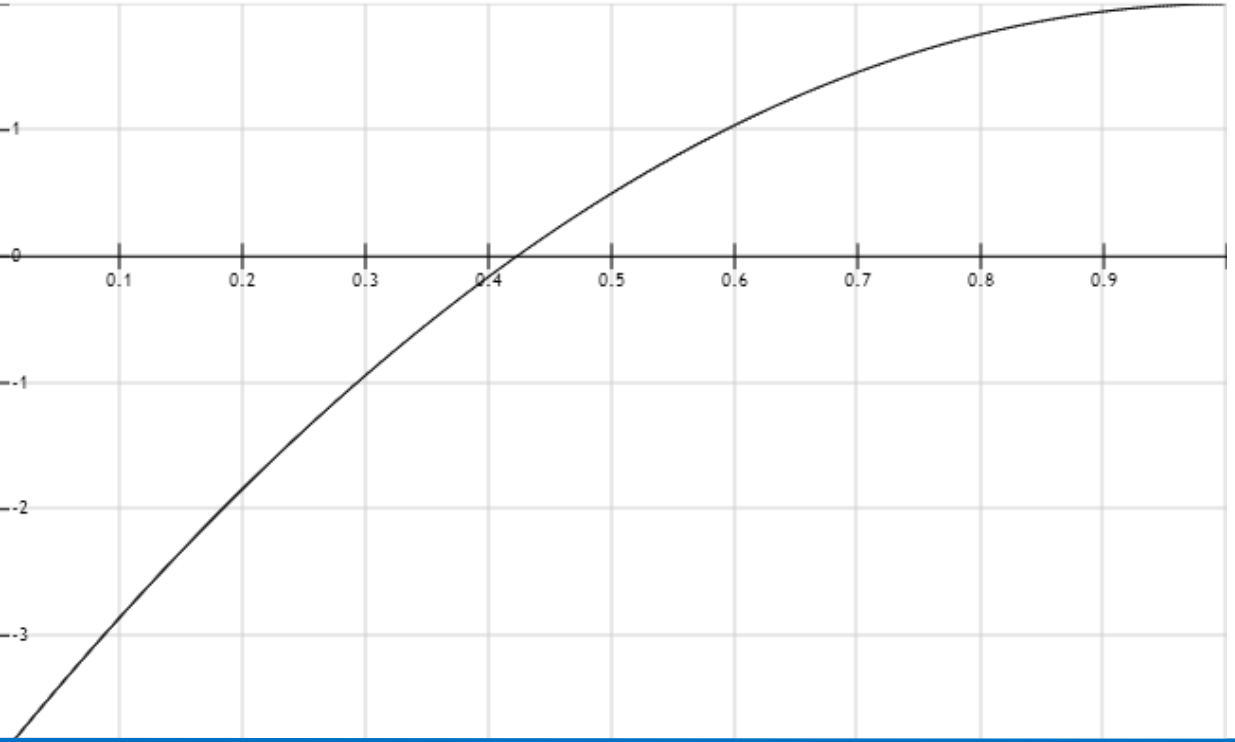
- Simulates the image of artifact of real-world camera, producing an immersive environment during the races.
- This effect is achieved by composing the image with a blurred and brightness filtered copy of itself.

GT Racing 2 Intel Special Effects – Bloom

- First step is to take the original framebuffer and apply a bright pass filter
- This will result in the parts that will have their white color enhanced

GT Racing 2 Intel Special Effects – Bloom

- The high filter pass uses an approximation formula, that allows only bright colors to pass:



$$f(x) = (-3 * (x-1)^2 + 1) * 2$$

GT Racing 2 Intel Special Effects – Bloom

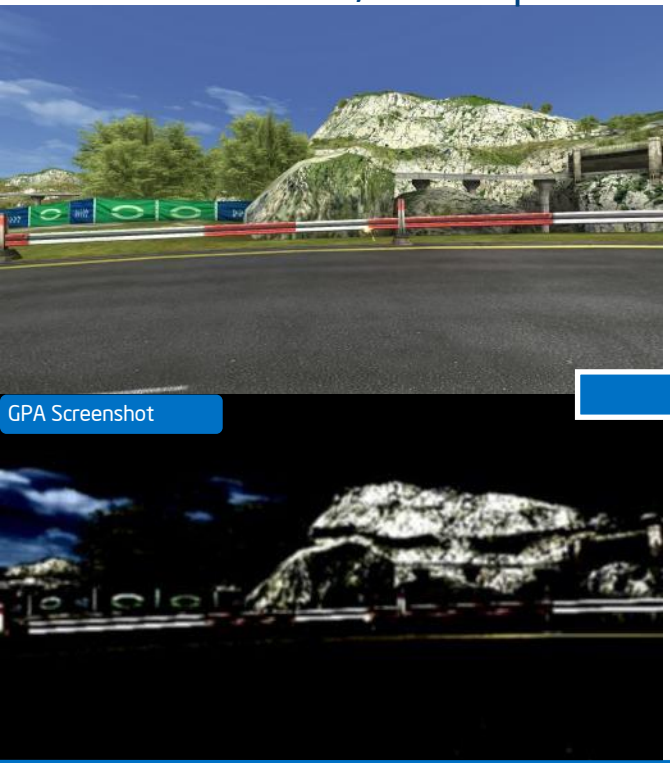
- Second step, is to apply an horizontal and then a vertical blur
- The bright pass filter output is used as input for the blur part

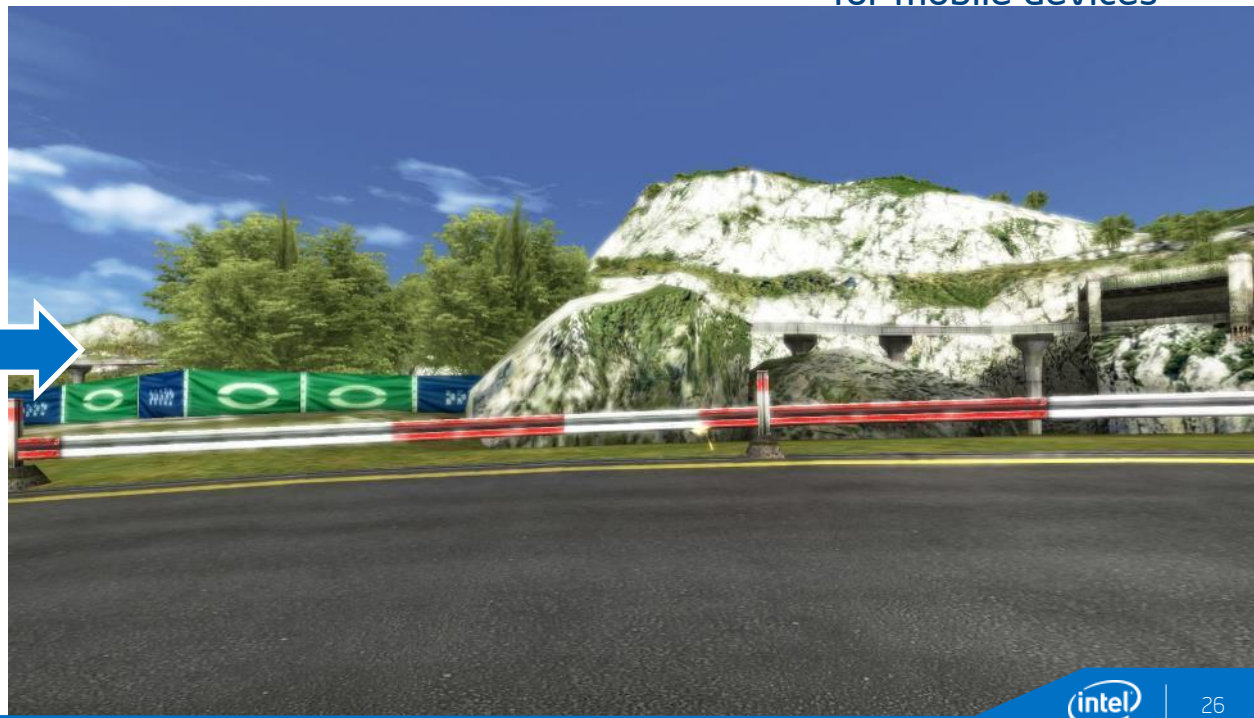
1st Blur – Horizontal Blur

2nd Blur – Vertical Blur

GT Racing 2 Intel Special Effects – Bloom

- In the end, we compose the blur output with the initial framebuffer, with a low-enough cost for mobile devices

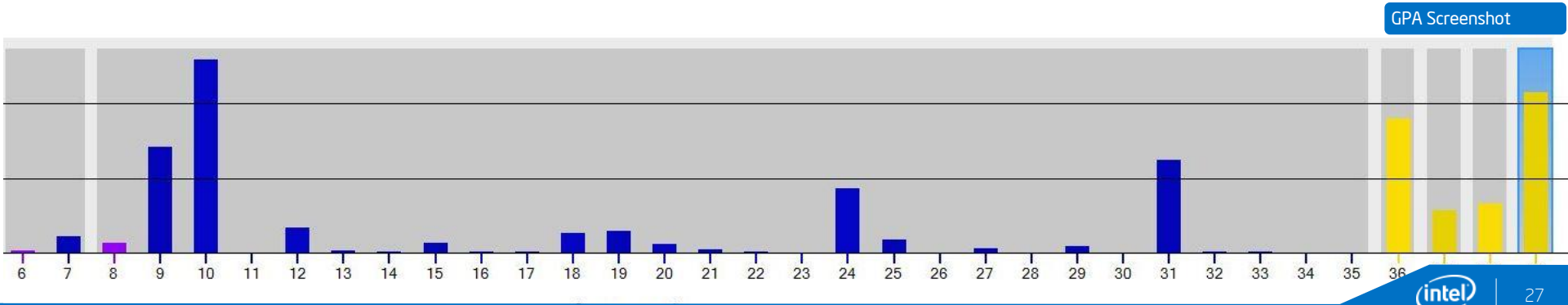




GT Racing 2 Intel Special Effects – Bloom

Bloom prost-processing effect cost

- Bright-pass filter: 1.4ms
- Horizontal blur pass: 0.57ms
- Vertical blur pass: 0.67ms
- Final compose, bloom: 2.17ms
- **Total**: 4.81ms

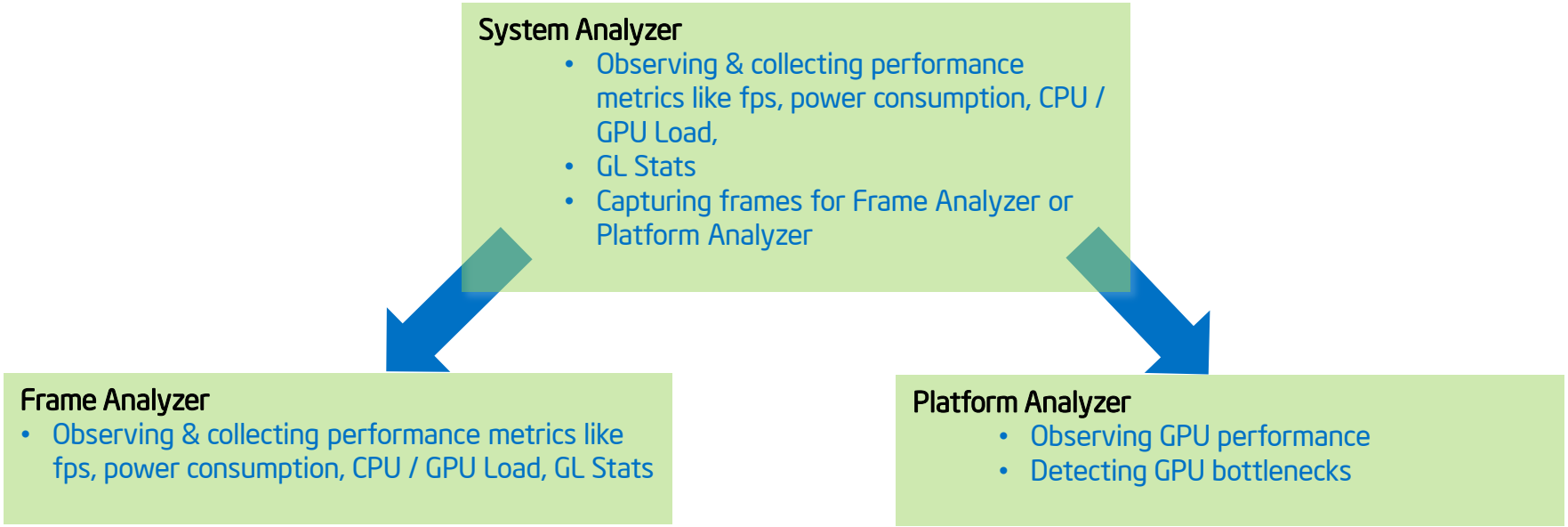


GT Racing 2 Intel Special Effects

Baytrail: Cutting Edge HW!

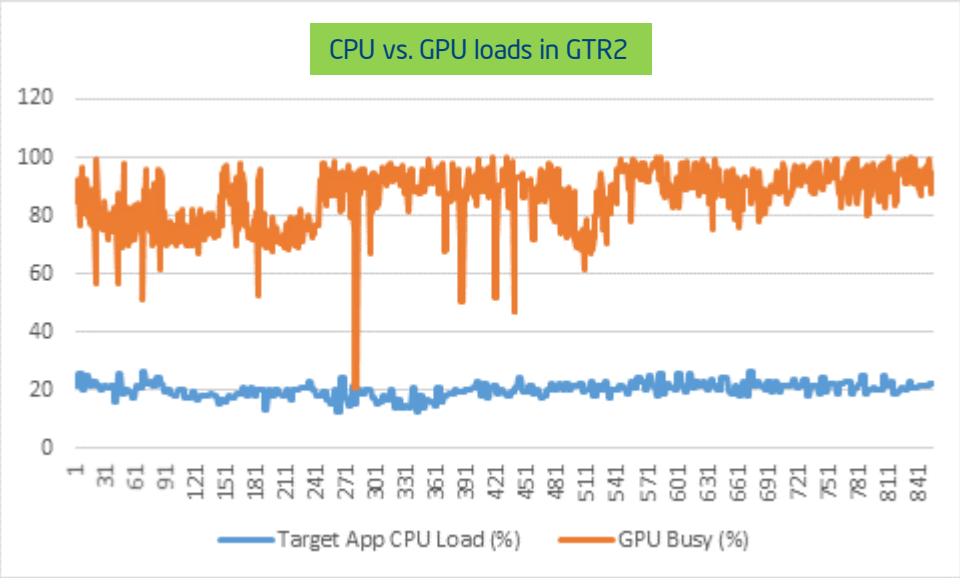
Optimization opportunities

Tools used to optimize GTRacing 2



Optimization opportunities

Step 1: Compare CPU and GPU load with System Analyzer



Observations:

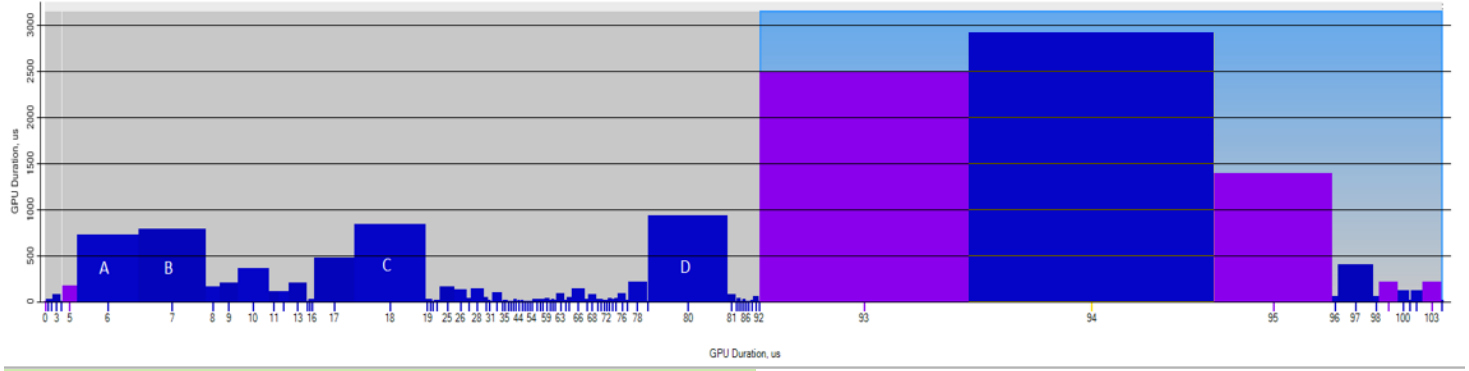
- GPU load around 90+ percent
- Fairly low CPU load
- Application is clearly GPU bound
- Most performance benefit can therefore be found in GPU pipeline.
- Proceed with Frame Analyzer

Activity:

- Dumped csv files of real time metrics ("App CPU Load %" and "GPU Busy %") from System Analyzer
- Loaded into Excel to present graph

Optimization opportunities

Pipeline Issues identified with GPA



Watch for unnecessary glClear calls.

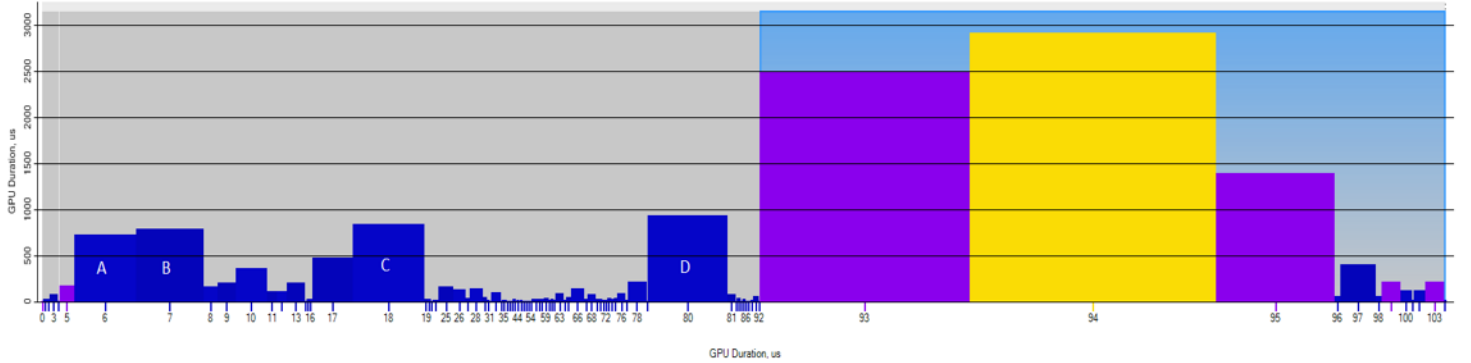
- Render targets on tablet devices are large, so calls to glClear() can be expensive.
- Very easy to leave unnecessary RT clears in the pipeline (purple ergs).
- GPA Identified about 5ms worth of RT clears which could be removed.

Activity:

- Dump frame from System Analyzer and open in Frame Analyzer
- Clear calls show up as purple on erg graph
- I use GPU Duration on both graph axes to really make these stand out

Optimization opportunities

Pipeline Issues identified with GPA



Big ergs are always worth look at :

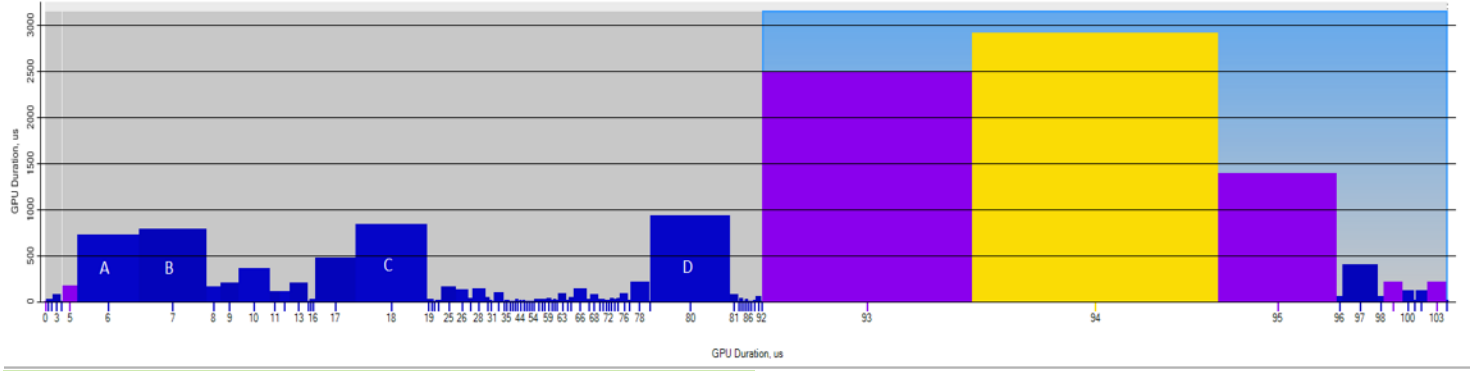
- Game was originally rendered half size then up-sampled (yellow erg)
- Very expensive process, almost worth rendering game full size instead – which in fact we ended up doing.

Activity:

- Dump frame from System Analyzer and open in Frame Analyzer
- Go for the largest erg's. See them as low hanging fruit. Identify erg function from shaders, geometry etc. and make judgment call.
- I use GPU Duration on both graph axes to really make these stand out

Optimization opportunities

Pipeline Issues identified with GPA



Not all big ergs are wrong ergs:

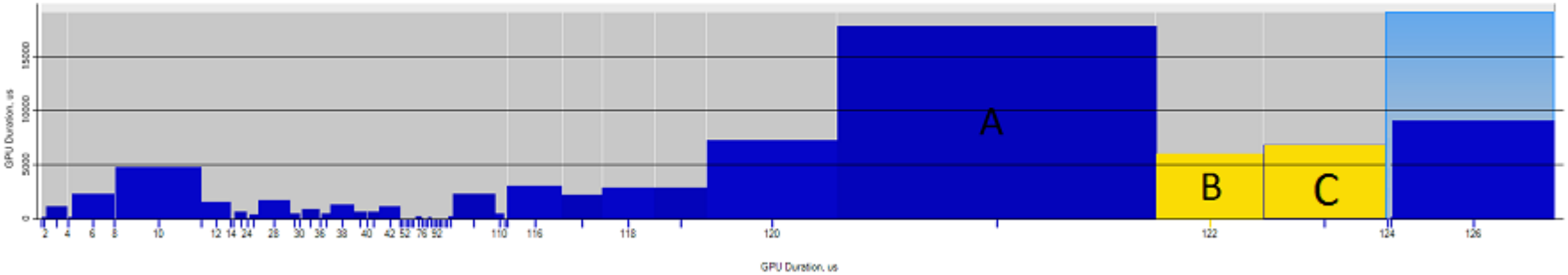
- Rendered objects A, B, C, and D are very expensive
- However, these are cars, and are key to the game
- Great example of spending cycles on the bits that matter in a frame

Activity:

- Dump frame from System Analyzer and open in Frame Analyzer
- Select erg and examine textures or Geometry to identify object rendered by erg

Optimization opportunities

Pipeline Issues identified with GPA



Its worth looking at small ergs too:

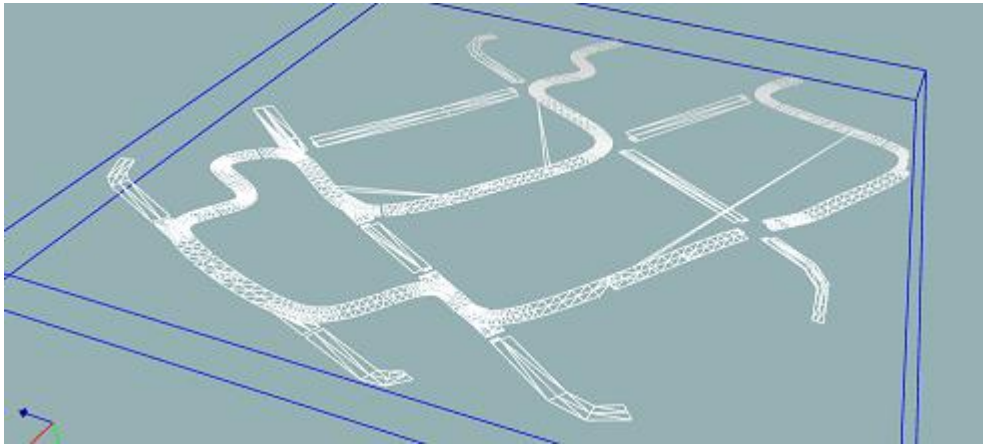
- Rendered objects B and C, are blur passes on data for effects, I expected lower cost for these
- Closer look showed these were actually full size RT's
- Reducing the RT size to ¼ native resulted in 2-3ms saving on frame time.

Activity:

- Dump frame from System Analyzer and open in Frame Analyzer
- Examine Geometry and textures to deduce erg action

Optimization opportunities

Pipeline Issues identified with GPA



Model View in GPA Frame Analyzer

Input-Assembler	
Primitive Count	1,958.0
Vertex Count	5,874.0
Vertex Shader	
VS Invocations	2,606.0
VS EU Active %	0.0
VS EU Stall %	0.0
Rasterizer	
Clipper Invocations	1,958.0
Post-Clip Primitives	1,096.0

Cut from GPA Frame Analyzer Details tab

CPU vs. GPU clipping

- Track render shows no CPU clipping
- All primitives from track model are sent to clipper
- All 1958 prims are put thru
- Almost 1K prims are clipped

Activity:

- Dump frame from System Analyzer and open in Frame Analyzer
- Examine Geometry and Details tab to see stats

- Clipping models on CPU would save GPU cycles
- Unfortunately, clipping not possible in the pipeline
- One that got away, but logged for next time.

Optimization opportunities

Pipeline Issues identified with GPA

Sometimes a fresh eye can help:

- Some observations suggested bloom effect was “washed out”
- Investigation showed that the bloom math was overly complex
- And it was loading the render target

What we suggested:

- Replacing bloom with simpler algorithm
- Using additive blend mode instead of loading the RT to alter it
- Prototyped in GPA!

Activity:

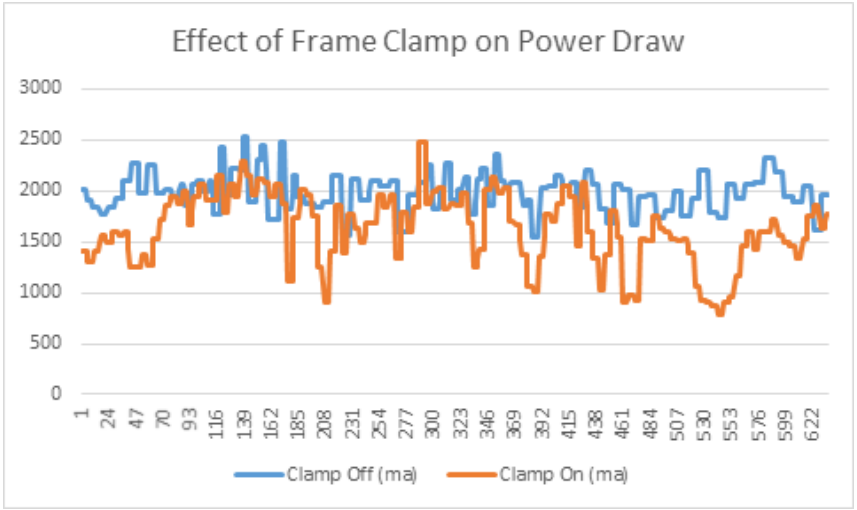
- Dump frame from System Analyzer and open in Frame Analyzer
- Find shader responsible for effect
- Edit shaders to experiment with effects ***without recompiling the game***!

Optimization opportunities

Power consumption: Frame clamp can be your friend:

Why looking at power draw is important:

- Improves available game play time
- Longer times between charging
- Fewer complaints – no one likes apps that drain the battery
- ***Save the Planet!***



Activity:

- Dump CSVs of Current or Power discharge from System Analyzer
- Load into excel & make a graph
- ***Easy really!***

Adding x86 Build Target to your Android Game

- Not very different to ArmV7a
  - 32 bit word size
  - Little-endian storage
  - HW FPU
  - Not usually anything to do about textures
- Minor differences
  - Need to watch alignment (aligned vs packed).
  - Any low level vector math needs translating (NEON to SSE)
  - Need to specify tool chain in Application.mk
  - Easy runtime and compile time checks to detect platform if you need them
  - Compiler flags (at O2)
    - -march=atom
    - -mssse3
    - -finline-limit (about 300 is good for x86)
- Common starting issues
  - Prebuilt libs will need recompiling
  - Textures may need converting
- In all – not too bad a job!

Summary:
Optimization is the key to "Next Level" Graphics on Mobile devices!

- Need high frame rate to allow room for effects like the ones we've seen.
- A 5ms effect means you need to shrink render time at 30fps by 15% to fit it in.
- Find those extra ms by profiling hard with GPA and staying on the look out for savings at all times.
- Remember: GPA is not just a PC tool any more
  - Get the latest at http://software.intel.com/en-us/vcsource/tools/intel-gpa

Ready for More? Look Inside™.

Keep in touch with us at GDC and beyond:

- Game Developer Conference
  Visit our Intel® booth #1016 in Moscone South
- Intel University Games Showcase
  Marriott Marquis Salon 7, Thursday 5:30pm
  RSVP at **bit.ly/intelgame**
- Intel Developer Forum, San Francisco
  September 9-11, 2014
  intel.com/idf14
- Intel Software Adrenaline
  @inteladrenaline
- Intel Developer Zone
  software.intel.com
  @intelsoftware