Geekbench 3 Workloads

Primate Labs

August 8, 2013

The goal of Geekbench is to provide an objective measure of computer hardware performance across different software platforms. It does this by executing 29 tests, or "workloads," each of which focuses on a particular aspect of a computer's performance. The workloads are divided into three categories: integer, floating point, and memory. The integer and floating point workloads test CPU performance: how fast a computer can perform calculations. The memory workloads test memory access performance: how fast a computer can retrieve a piece of data from memory.

1 Integer Workloads

The integer workloads measure how fast a computer can execute operations on integers. To focus on CPU performance, there is no file input or output in the timed sections of the integer workloads. Additionally, each workload is analyzed to reduce memory accesses and eliminate memory access bottlenecks. The integer workloads in Geekbench 3 are explained in this section.

1.1 AES

The advanced encryption standard (AES) defines a symmetric key block encryption algorithm. AES encryption is used in security tools such as SSL, IPsec, and GPG. The algorithm encrypts a message by executing a sequence of reorderings and substitutions in 16-byte blocks. These operations make heavy use of lookup table references and bit-level operations such as shifting, bitwise and, and bitwise xor. The AES workload in Geekbench uses a hardcoded 128-bit key aligned in memory to 16 bytes. The clear text input to AES on desktop computers is a 32 MB string and on mobile platforms is an 8 MB string. Some machines provide the AES-NI instructions for fast AES execution. The AES workload in Geekbench uses these instructions when they are available.

1.2 Twofish

Twofish is a symmetric key block encryption algorithm. It is a member of the family of encryption algorithms know as Feistel Cipher and is included in the OpenPGP standard. The twofish implementation in Geekbench uses a hardcoded 128-bit encryption key. The input is a generated text of 32 MB on desktop computers and 8 MB on mobile platforms.

1.3 SHA1 and SHA2

SHA1 and SHA2 are cryptographic hash algorithms. Given an input of arbitrary length, a cryptographic hash algorithm computes a fixed length “digest” of the message. The algorithm is designed such that the digest is easy to compute and it is impractical to find an input that generates a prescribed digest. Furthermore, small changes in the input should produce significant changes in the digest. Cryptographic hash algorithms are used to store passwords and verify the authenticity of data.

SHA1 was the NIST standard cryptographic hash algorithm until approximately 2010 when it was replaced by SHA2 due to weakness against collision attacks. SHA2 processes a message using a block structure similar to that of SHA1, but with more sophisticated computation at each round. Collision attacks against SHA1 remain impractical, however, and it still in widespread use [1]. The SHA1 workload in Geekbench computes a 160-bit digest on a generated text message. The message length is 64 MB on desktop machines and 16 MB on mobile devices. The SHA2 workload computes a 256-bit digest for a generated message. The message length is 32 MB for desktop machines and 8 MB for mobile devices.

1.4 BZip2 Compression and Decompression

BZip2 is a file compression algorithm. It uses Huffman coding and the Burrows-Wheeler transform. The BZip workloads in Geekbench compress or decompress an HTML-formatted ebook from Project Gutenberg using bzlib version 1.0.6. On a desktop machine, the input is “Ulysses” by James Joyce. The HTML file is 1823 KB uncompressed and 513 KB compressed. On a mobile device, “Dubliners” by James Joyce is used. It is 458 KB uncompressed and 116 KB compressed.

1.5 JPEG Compression and Decompression

The JPEG lossy image compression algorithm encodes an image in 8×8 blocks. Each block is transformed using a discrete cosine transform (DCT) and each DCT coefficient is quantized to achieve compression. The coarseness of the quantization is determined by the desired quality of the compressed image. The input image for the JPEG compression and decompression workloads is the 24-bit 3-channel image shown in Figure 1. A 6.4 megapixel image is used for desktop workloads and a 1.6 megapixel image is used for mobile workloads. For the compression workload, the quality parameter is set to 90/100.

1.6 PNG Compression and Decompression

The PNG lossless image compression algorithm encodes an image in two stages. It transforms the image using a prediction/correction scheme then compresses the resulting sequence using the Deflate algorithm. Deflate uses Huffman coding and the LZ77 algorithm. The Geekbench workload compresses images using libpng 1.6.2. The input image for the PNG compression and decompression workloads is the 24-bit 3-channel image presented in Figure 1. For the compression workload, the image is scaled to 0.4 megapixel on desktop platforms and to 0.1

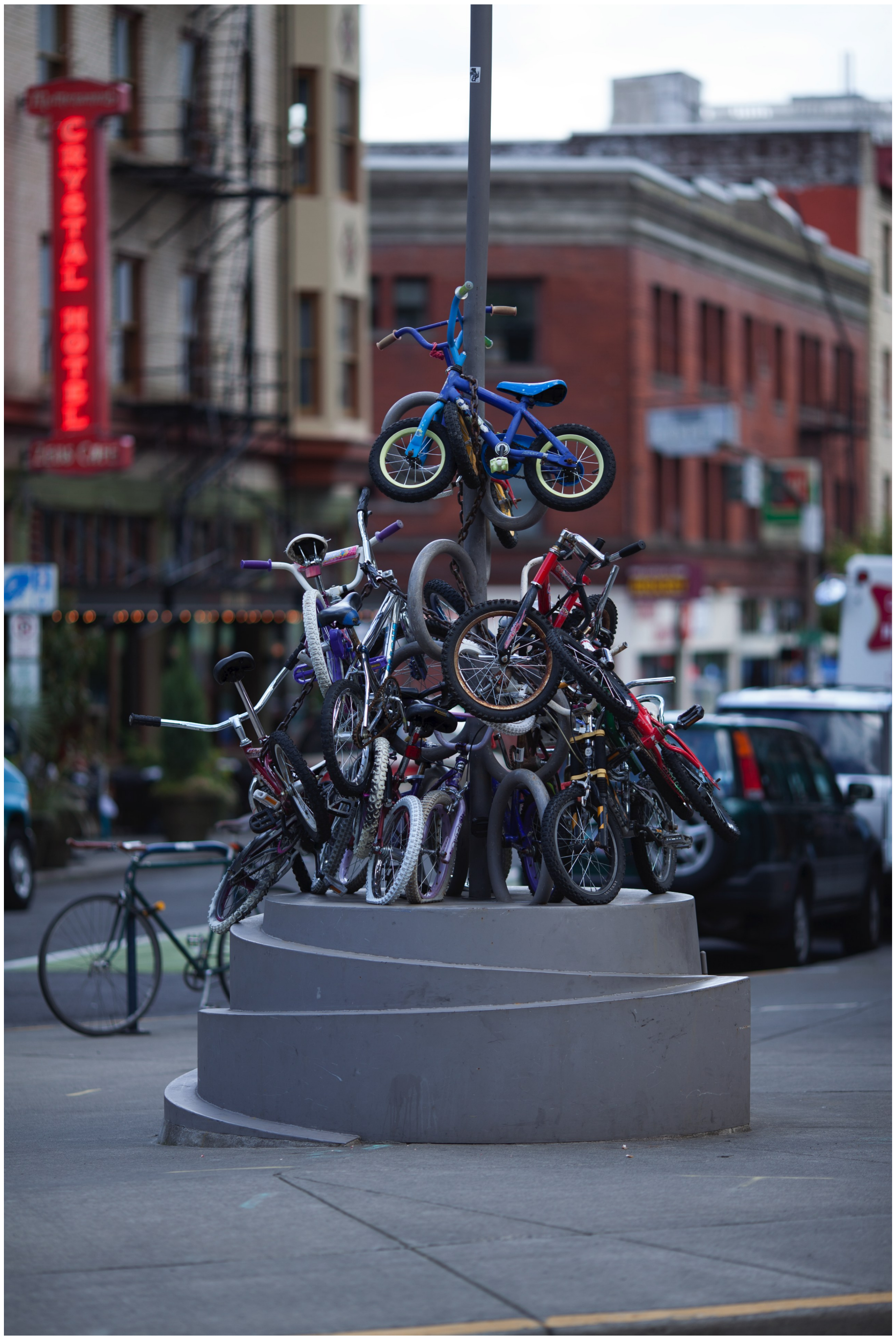

Figure 1: Input image for image workloads

megapixel on mobile. For the decompression workload, the image is scaled to 2.5 megapixel on desktop plagforms and 0.6 megapixel on mobile devices.

1.7 Sobel

The Sobel operator computes an approximation of the norm of a smoothed gradient of an image taken as a 2D function. The value of the operator is large on sharp edges of the input image, so it useful in edge detection. The Sobel operator is one component of the Canny edge detector. The Sobel implementation in Geekbench convolves the input image with the kernel:

1	0	-1
2	0	-2
1	0	-1

to obtain the x component of the smoothed gradient. It uses the kernel:

1	2	1
0	0	0
-1	-2	-1

to obtain the y component. The implementation computes an approximation of the magnitude of the smoothed gradient as the sum of the absolute values of the components: $|[x, y]^\top| \approx |x| + |y|$. The Sobel workload uses integer multiplication and addition operations heavily. The test image for the Sobel workload is 10 mexapixel on desktop machines and 4.9 megapixel on mobile devices. The input image is a 24-bit 3-channel image and is converted to grayscale as part of the workload. The input image for this workload is shown in Figure 1. The output from the Sobel operator is presented in Figure 2.

1.8 Lua

The Lua workload tests execution of a compiled scripting language on a virtual machine. The script is based on the page generation code in the Geekbench browser. It is written in Lua and uses dkjson to parse a JSON string containing Geekbench results. The script iterates over the results and generates two tables: one for results from 32-bit machines and one for results from 64-bit machines. It sorts both tables by score and returns the CPU information for the row with the maximum score in the 32-bit table. The JSON input is 333 KB on desktop platforms and 169 KB on mobile platforms. The same Lua script is used on both desktop and mobile devices.

1.9 Dijkstra

Dijkstra's algorithm computes single-source shortest paths in weighted graphs. Some applications include routing of computer network traffic and route calculation in maping systems such as OpenStreetMap or Google Maps. Input for the Dijkstra workload in Geekbench is a graph representing the streets and roads in the Waterloo region in Ontario, Canada. The

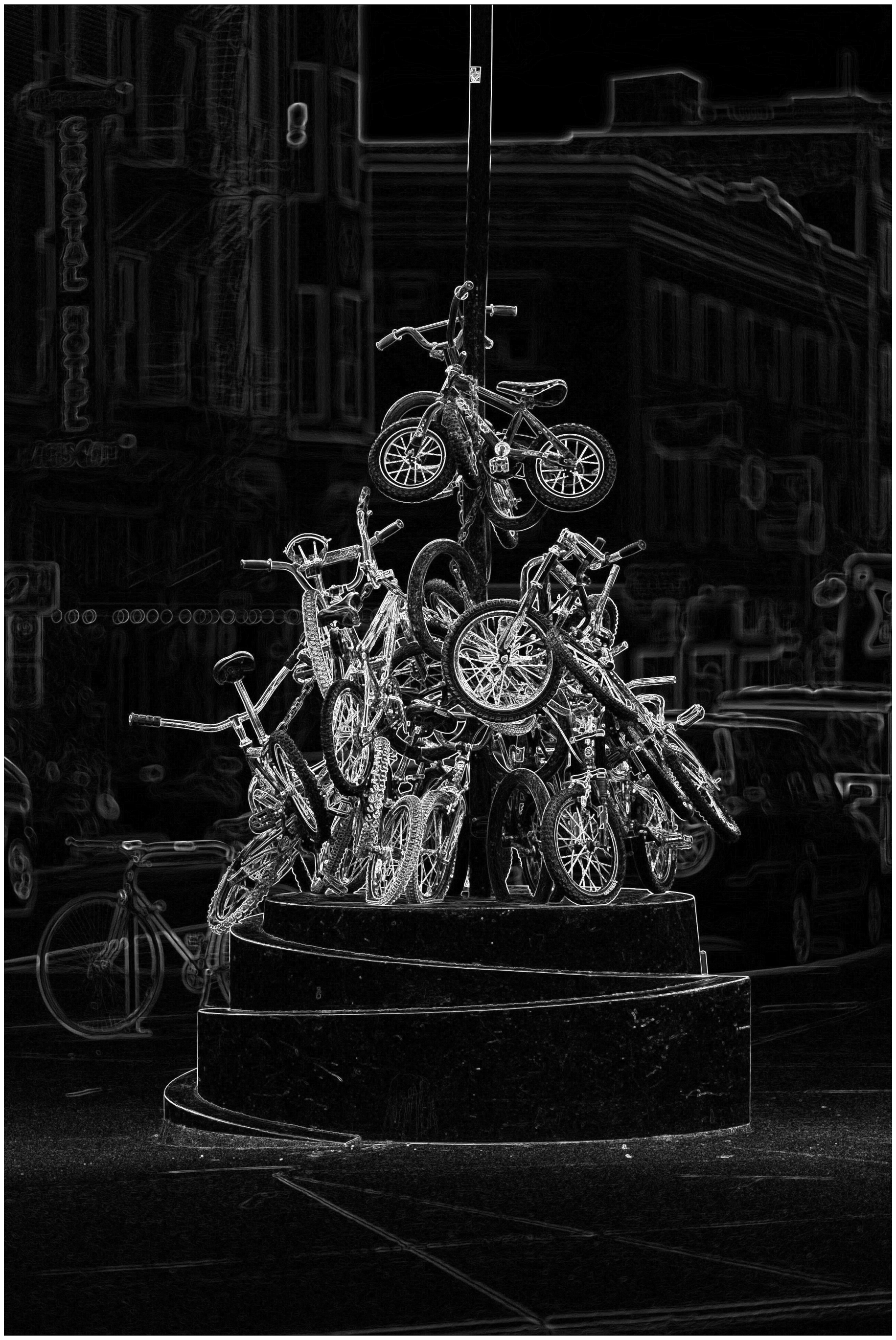

Figure 2: Output from the Sobel workload

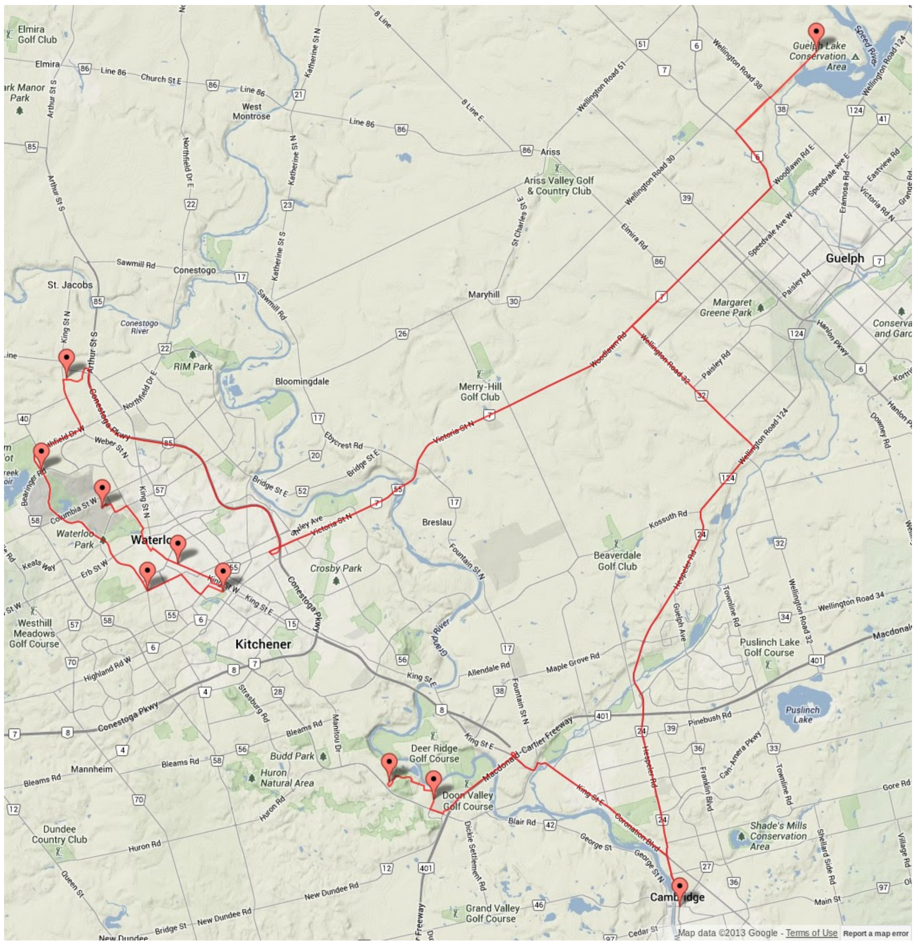

Figure 3: The routes computed by the Dijkstra workload

data was extracted from OpenStreetMap. The graph contains 79392 nodes and 162644 edges with integer weights approximating travel time along the section of road represented by the edge. The route includes a source and 9 ordered destinations as shown by the markers in Figure 3.

On desktop machines, the shortest path between each pair of destinations on the route is computed. On mobile devices, only the first three sections of the route are computed.

The computation in the Geekbench implementation of Dijkstra's algorithm is dominated by operations in a Fibonacci queue. This tests a machine's performance on data structures and pointer dereferences.

2 Floating Point Workloads

The floating point workloads measure computer performance on floating point operations. To focus on CPU performance, there are no file input or output operations in the timed sections of the workloads. The workloads are analyzed to remove memory bottlenecks and ensure that execution times are bound by CPU performance and not memory access performance.

2.1 Black-Scholes

The Black-Scholes stochastic partial differential equation models option pricing on financial markets. For some special cases, such as European call and put options, the equation can be solved exactly. The solution for European call and put options is called the Black-Scholes formula. The Black-Scholes workload in Geekbench calculates option prices using the Black-Scholes formula. It uses floating point computation intensively and has a low memory bandwidth requirement, so it is an effective measure of floating point performance. The formula uses expensive functions from the standard math library: `logf`, `sqrtf`, and `expf`. The workload uses a spot price of \$42, a strike price of \$40, and a risk free rate of 0.1. The price is computed over 3 million time steps on desktop machines and 500,000 time steps on mobile devices.

2.2 Mandelbrot

The Mandelbrot set is the set of points z_0 in the complex plane such that $|z_n|$ is bounded in the recurrence

$$z_{n+1} = z_n^2 + z_0, n = 0, 1, ...$$

as $n \to \infty$. Computing a numerical approximation of the Mandelbrot set involves floating point addition and multiplication in a loop to verify the boundedness of the recurrence for each point in the domain of interest. The Mandelbrot workload in Geekbench computes the approximation over the region of the complex plane: $\{r + ci; -1.5 \leq r \leq 1.5 \text{ and } -1 \leq c \leq 1\}$. The region is discretized by a uniform grid of 800×800 for desktop machines and 400×400 for mobile platforms. The workload computes 255 iterations of the recurrence for each point z in the grid. If $|z_n| \geq 2$ at any point during computation of the recurrence then the recurrence is assumed to be unbounded and the loop for that point is terminated. The Mandelbrot set computed by Geekbench is illustrated in Figure 4.

2.3 Sharpen Image

The sharpen image workload implements an unsharp mask filter using the negative of the Laplacian operator. The operation is implemented in Geekbench by convolving in the spatial domain each channel of the image with the 3×3 kernel:

0	-1	0
-1	5	-1
0	-1	0

All addition and multiplication operations in the convolution are floating point. Additionally, the workload performs pointer dereferences to compute input and output pixel addresses. The input image for the sharpen workload is the 24-bit 3-channel shown in Figure 1. The image is scaled to 6.4 megapixel on desktop platforms and 1.6 megapixel on mobile devices. The output from the sharpen workload is shown in Figure 5

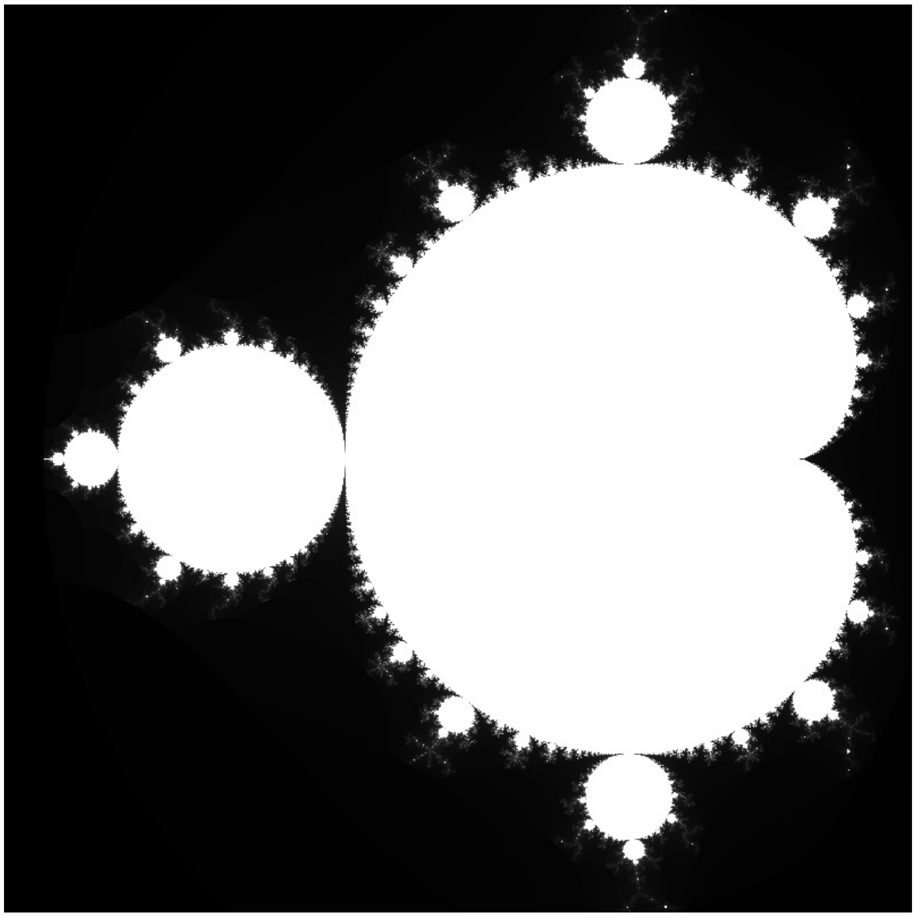

Figure 4: The set computed by the Mandelbrot workload

Figure 5: Output from the Sharpen workload

2.4 Blur Image

The blur image workload blurs each channel of an input image by convolving it with a 5×5 Gaussian kernel in the spatial domain:

1	4	6	4	1
4	16	24	16	4
6	24	36	24	6
4	16	24	16	4
1	4	6	4	1

All addition and multiplication operations in the convolution are floating point. This workload tests performance on floating point operations interleaved with memory reads and writes. The sharpen image workload also computes a spatial convolution, but blur performs more floating point operations and memory accesses per pixel due to its larger kernel size. The input image for the blur image workload is 2.5 megapixel on desktop platforms and 0.6 megapixel on mobile platforms. Both are 24-bit RGB resized versions of the image shown in Figure 1. The blurred image computed by the workload is presented in Figure 6

2.5 SGEMM and DGEMM

The general matrix multiplication (GEMM) workloads accept three $N \times N$ matrices A, B, and C and compute the result:

$$C = AB + C.$$

The GEMM algorithm divides the matrices into blocks and operates on one block at a time. This improves cache coherency and leads to faster execution compared to an implementation of the "pencil and paper" matrix multiplication algorithm. In Geekbench, SGEMM performs the operation using single precision floating point and DGEMM performs it using double precision floating point. The block sizes are empirically determined values that give good cache performance across tested platforms. SGEMM uses a block size of 128×128 elements and DGEMM uses 64×64 elements. The input values are $A_{ij} = B_{ji} = (i + jN)(\mod 10)$ and $C_{ij} = 0$. For both SGEMM and DGEMM, $N = 896$ on desktop machines and $N = 512$ for mobile devices.

2.6 SFFT and DFFT

The Fourier transform decomposes an input signal into a linear combination of a basis of trigonometric polynomials. In other words it transforms an array into a sum of sines and cosines of various frequencies. The fast Fourier transform (FFT) algorithm executes this operation efficiently. The Geekbench implementation is a recursive Cooley-Tuckey radix-2 decimate in time FFT. It performs the bit-reversed index reordering step before the recursion and contains special case processing for array lengths of 2, 4, 8, and 16. The special cases contain fewer branching and array dereference operations than the general case recursive function. Finally, the workload precomputes the complex exponentials (the "twiddle factors") and does not count the computation of these values in the workload execution time.

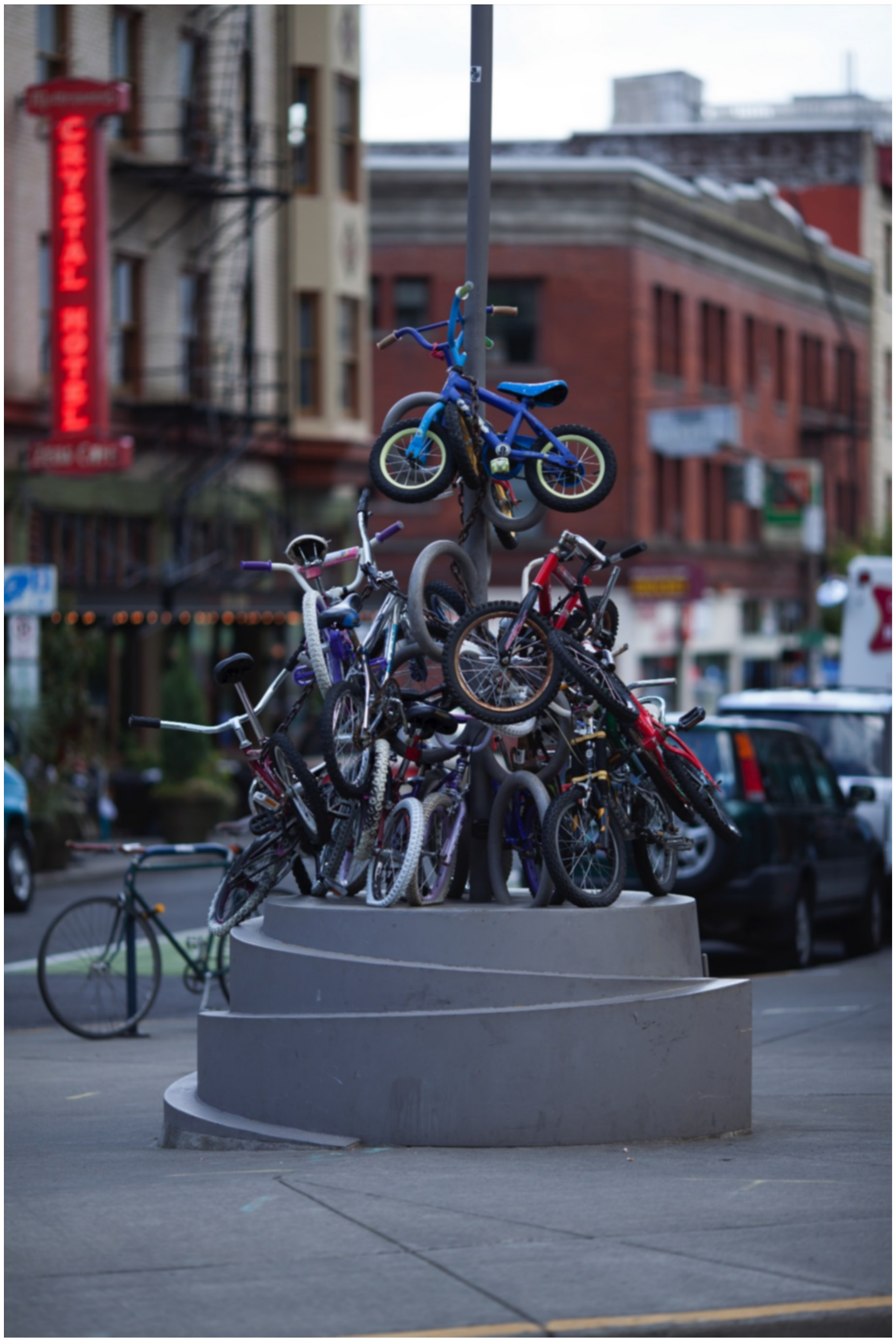

Figure 6: Output from the Blur workload

The workload simulates a frequency analysis in an audio processing application. A 1D input array is partitioned into chunks of 4096 elements and an FFT is computed for each chunk. The SFFT workload performs this computation using single precision and the DFFT workload performs it using double precision. On desktop platforms, the input array contains approximately 8 million data points ($8{\cdot}1024{\cdot}1024$) giving a 33.5 MB input array for SFFT and a 67 MB input array for DFFT. On mobile devices, the input array contains approximately 2 million data points ($2 \cdot 1024 \cdot 1024$) giving an 8.3 MB input array for SFFT and a 16.7 MB input for DFFT.

2.7 N-Body

The N-body workload computes a 3D gravitation simulation using the Barnes-Hut method. To compute the exact gravitational force acting on a particular body x in a field of N bodies requires $N - 1$ force computations. The Barnes-Hut method reduces the number of force computations by approximating as a single body any tight cluster of bodies that is far away from x. It does this efficiently by dividing the space into octants—eight cubes of equal size—and recursively subdividing each octant into octants, forming a tree, until each leaf octant contains exactly one body. This recursive subdivision of the space requires floating point operations and non-contiguous memory accesses.

With the space divided, the force exerted on a body x by all bodies in an octant can be approximated by the force exerted on x by the centre of mass of the octant. This approximation is used when the side length l of the octant is small relative to the distance d from x to the center of mass of the octant: if $l/d < T$ for a fixed threshold T. The Geekbench N-body workload uses $T = 0.5$. Once the force has been computed for all bodies in the simulation, the position of each body is updated. This step involves iterating over the bodies in storage order and solving a system of first-order ordinary differential equations for each. This is done using the forward Euler method with a time step of 500. The workload simulates gravitation for 1000 bodies of 1000 Kg each. The bodies are initially placed at the gridpoints of a uniform grid in the computation domain. The workload simulates 40 time steps on desktop machines and 5 time steps on mobile devices.

2.8 Raytrace

In a ray tracing rendering system, a 3D scene is described by a mathematical model. The scene is then rendered by choosing a camera position in scene and placing the image plane between the camera and the objects to be rendered. A ray is passed from the camera lens through each pixel in the image plane and into the scene. When a ray intersects an object, the material properties of the object at the point of intersection and the light source in the scene determine the color of the corresponding pixel in the image plane. The ray trace workload renders a 3D scene using a software ray tracer on the CPU. It simulates shading, reflection, and diffusion effects. On Desktop machines, the rendered image is 0.36 megapixels and on mobile platforms it is 0.18 megapixels. The rendered scene is presented in Figure 7.

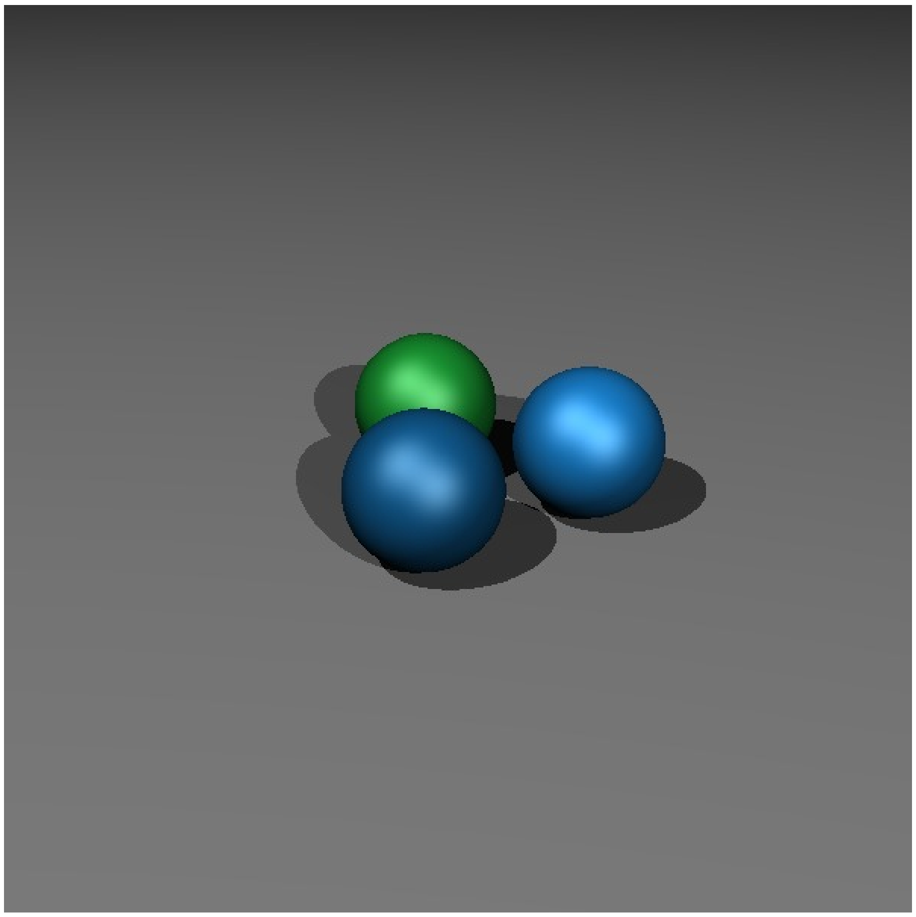
Figure 7: Output from the Raytrace workload

3 Memory Workloads

The memory workloads in Geekbench gauge performance of the memory subsystem under sustained load by sequentially accessing and manipulating large regions of memory. These workloads are based on the STREAM benchmarks by John D. McCalpin.

3.1 Stream Copy

This workload executes a value-by-value copy of an array of singe precision floating point values from one memory region to a disjoint region:

```
b[i] = a[i] for i = 1, 2, ..., n.
```

The array length is approximately forty million elements ($40 \cdot 1024 \cdot 1024$) on desktop platforms and approximately five million ($5 \times 1024 \times 1024$) on mobile devices giving a copy of 168 MB on desktops and 21 MB on mobile.

3.2 Stream Scale

The stream scale workload multiplies by a constant each value in an array of single precision floating point number and writes the result to an array of the same type in a disjoint region of memory. The values are chosen such that no overflow occurs:

```
b[i] = k*a[i] for i = 1, 2, ..., n.
```

The array length is approximately forty million elements ($40 \cdot 1024 \cdot 1024$) on desktop platforms and approximately five million ($5 \cdot 1024 \cdot 1024$) on mobile devices giving a copy of 168 MB on desktops and 21 MB on mobile.

3.3 Stream Add

The stream add workload reads 32-bit single precision floats from two non-overlapping arrays, adds them, and writes the result to a third array of the same type. The values are chosen such that no overflow occurs:

```
c[i] = a[i] + b[i] for i = 1, 2, ..., n.
```

The array length is approximately forty million elements ($40 \cdot 1024 \cdot 1024$) on desktop platforms and approximately five million ($5 \cdot 1024 \cdot 1024$) on mobile devices giving intputs of 168 MB on desktops and 21 MB on mobile.

3.4 Stream Triad

The stream triad workload combines stream addition with scalar multiplication. It sequentially reads single precision floating point values from two non overlapping arrays, multiplies one of the values by a constant and adds the result to the other value. The result of this addition is written to the corresponding location in a third array of the same type:

```
c[i] = k*a[i] + b[i] for i = 1, 2, ..., n.
```

The array length is approximately forty million elements ($40 \cdot 1024 \cdot 1024$) on desktop platforms and approximately five million ($5 \cdot 1024 \cdot 1024$) on mobile devices giving inputs of 168 MB on desktops and 21 MB on mobile.

References

[1] Bruce Schneier. When will we see collisions for sha-1? `http://www.schneier.com/blog/archives/2012/10/when_will_we_se.html` (retrieved August 8, 2013), 2012.