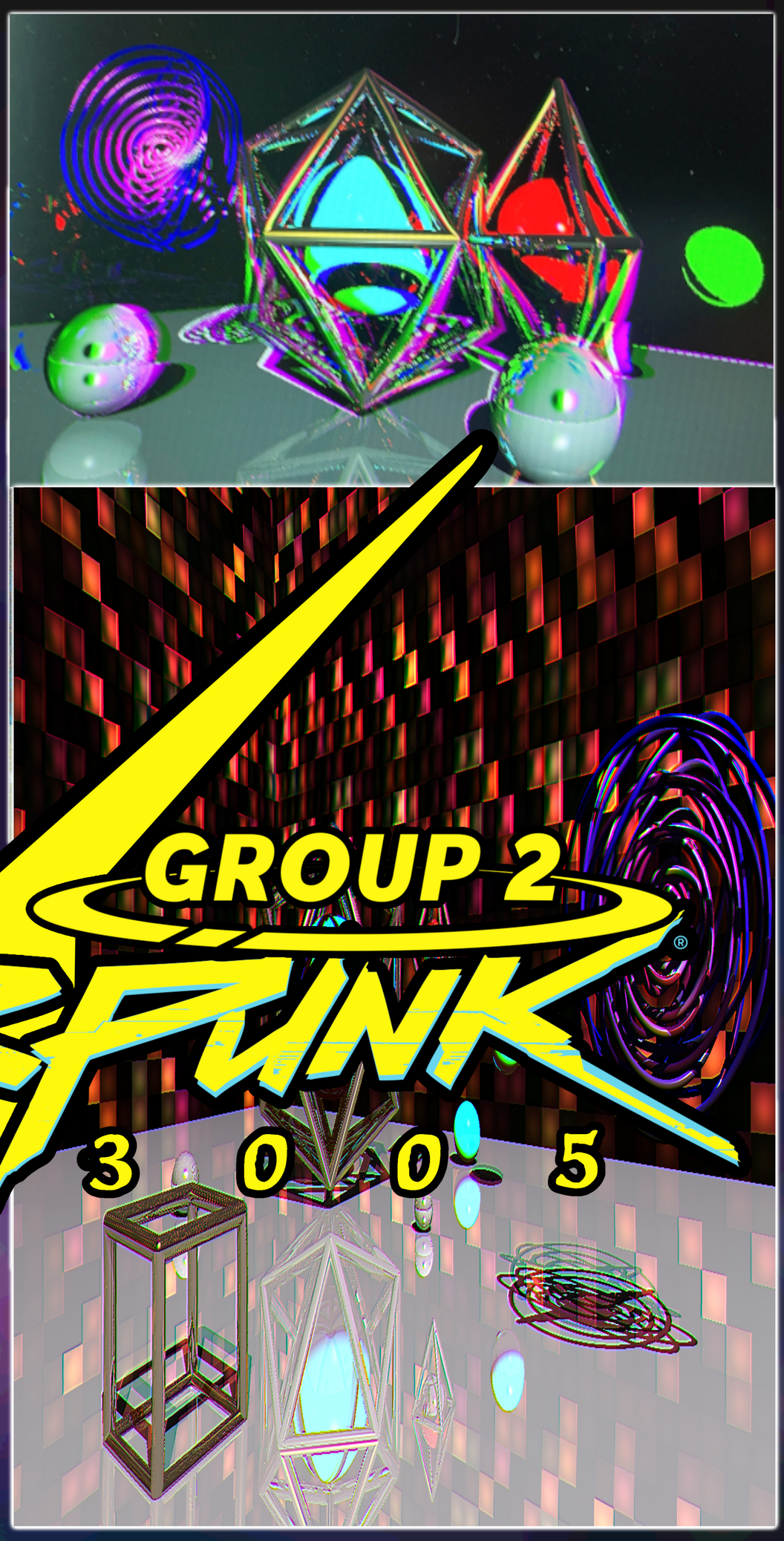
REAL-TIME RAY-TRACING ON GPU

FROM THEORY



rate of performance increase than CPUs. Modern CPU design is optimized for rapid execution of serial code. It is becoming increasingly difficult to realize performance benefits by adding extra transistors. The GPU on the other hand, is optimized for massively parallel vertex and fragment shading code . Transistors spent on additional functional units directly improve performance. As such, GPUs are able to utilize extra transistors more efficiently than CPUs, and GPU performance gains will continue to out pace CP... mance gains as semiconductor fabri... vances.

R... to expan... programmability has enabled several algorithms to be ported to the GPU [9, 30, 43, 46], many of which run at rates competitive with or faster than a CPU-based approach. The ubi... and low cost of graphics processors, coupled with t... formance on parallel applications, makes them ...active architecture for implementing realtime ray tracing.

Graphics algorithms like ray tracing can benefit from a GPU-based implementation in two other ways. First, when an algorithm executed on the GPU finishes running, the data meant for display is already on the graphics card. There is no need to transfer data for display. Second, graphics algorithms can work as hybrid algorithms, supplementing the capabilities of the GPU, and leveraging existing GPU rendering capabilities. For example, a ray tracer could be used to add global illumination effects like shadows, reflections, or indirect lighting to a polygon renderer.

We will examine two different approaches to using the GPU for ray tracing. Both utilize the high computational throughput of the GPU to obtain rendering rates compara... ble to those obtained by the fastest software-only ray tracers. Section 3.2 describes the work done by Carr et al. [10] in configuring the GPU as a ray-triangle intersection engine. Section 3.3 then describes the work by Purcell et al. [69] in mapping the entire ray tracing computation to a programmable GPU. Before discussing either implementation, we review the modern programmable graphics pipeline in Section 3.1.

3.1. Modern Graphics Pipeline

Figure 5 shows an abstraction of the graphics pipeline used by GPUs like the ATI Radeon 9800 Pro [3] or the NVIDIA GeForce FX 5900 Ultra [57]. The vertex and fragment stages are implemented with programmable engines that execute user defined programs. Modern GPUs have support for floating point computation throughout most of the pipeline, and have floating point frame buffer and texture memory. The

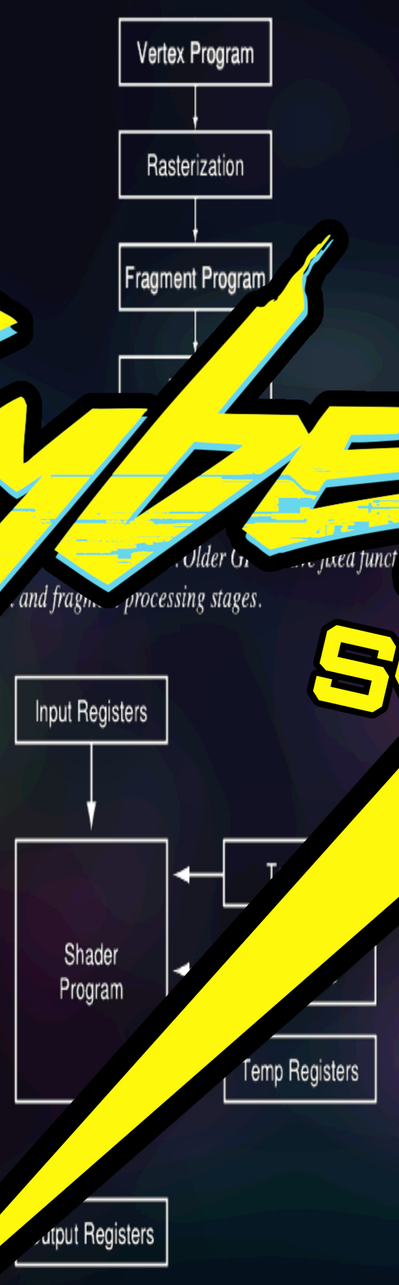


Figure 6: *The programmable fragment processor. A shader program can read data from input registers, constants, texture memory, and temporary registers. Temporary registers store intermediate calculations, and the output registers store the final color values for the fragment.*

The execution model for the fragment engine is shown in Figure 6. Fragment programs are written in a 4-way SIMD assembly language [60, 58], which includes common operations like add, multiply, dot product, and texture fetch. Fragment programs are currently limited to 64 or 1024 instructions, depending on the specific chip being used. These limits are likely to increase with future generations of GPUs.

Current GPUs do not allow data dependent looping or branching within fragment programs, though this limitation is likely to be removed in upcoming generations. Data de...

Our Cyberpunk-styled Shader:
https://www.shadertoy.com/view/mdjfRd#

TO REALITY

LIU TONGHE
LI ZENAN
SUN HAO

NUS National University of Singapore | Computing