The ComplexVision Manual

Ekobadd

May 6, 2024

1 About ComplexVision

This piece of software is a passion project of mine, a culmination of significant effort and research over years that allows the production of a variety of renders of the Mandelbrot set. Ultimately, it is intended to allow the user to define a path for a virtual camera to follow to produce an animation of renders over the complex plain, which is populated with samples from any one of the infinite family of fractals defined by the equation $Z_n = Z_{n-1}^2 + C$ (not just the eponymous Mandelbrot set).

It supports a variety of techniques for sampling the set, as well as for coloring and shading it. The default and most common such method uses a renormalized iteration count with solid-color interior and an exterior colored using a palette with monotonic cubic interpolation, but many methods exist which are enumerated in the following pages.

2 The Rendering Pipeline

The rendering pipeline consists of three steps.

First, the render context provides information necessary to produce a *camera matrix*, such a matrix determines the position and boundaries of the rendering frame in the complex plane. For an animation, many such matrices are generated at once before they are partitioned out among available computing resources.

Second, the matrix is used to generate a sample frame. The frame is populated with values which encode - in some fashion - some information about the sampling process. Typically, this is the renormalized iteration count.

Finally, this frame is converted into an image file by using the sample from to generate colors.

3 Implementation

3.1 Resource Acquisition and Utilization

ComplexVision will, by default, acquire almost as many computing resources as the OS will allow it to. This entails the creation of multiple CPU threads and the acquisition of all OpenCL-capable devices (typically graphics cards), all of which will be occupied primarily with performing Mandelbrot samples, as a typical animation often requires more than a trillion such samples. Additional threads handle ancillary operations, primarily saving to disk.

A substantial bottleneck for performance is memory. This is because multiple frames may be required for each concurrent render. For example, multiple colored frames may be produced from a single sample frame (a full-size one, a slightly cropped version to produce the next frame, and so on) but a frame often exceeds hundreds of megabytes in size, causing issues with allocating the space for all of them.

Therefore, a significant issue arises in the problem of keeping memory usage below a certain threshold while maximizing the utilization of available resources. Luckily, all of the most computationally complex issues are highly parallelizable. This means that individual frames can themselves be partitioned among multiple resources. However, memory remains the bottleneck in cases where the workload does not saturate these resources. A scarcity of memory may also force the host (but not the device) to wait for a save to disk to complete before rendering the next frame.

The render controller allocates new frames until available computing resources are saturated by the workload or until it reaches its allocation limit. Computing resources include all CPU cores and OpenCL-capable devices.