Virtual Intraoperative Cholangiogram Using WebCL

Alexander YU [a], Doga DEMIREL [ab1], Tansel HALIC [a,2], Sinan KOCKARA [a]
[a] *Department of Computer Science, University of Central Arkansas*
[b] *Department of Computer Science, University of Arkansas at Little Rock*

Abstract. In this paper, we propose a Virtual Intraoperative Cholangiogram (VIC) training platform. Intraoperative Cholangiogram (IC) is an imaging technique of biliary anatomy with using fluorescent fluids sensitive to the X-Rays. The procedure is often employed to diagnose the difficult cases such as abnormal anatomy or choledocholithiasis during the laparoscopic cholecystectomy. The major challenge in cholangiogram is accurate interpretation of the X-Ray image, which requires extensive case training. However, the training platforms that support generation of various IC cases have been lacking. In this study, we developed a web based platform to generate IC images from any virtual bile duct anatomy. As the generation of X-Ray image from 3D scene is a computationally intensive task, we utilized WebCL technology to parallelize the computation for achieving real-time rates. In this work, we present details of our WebCL IC generation algorithm and benchmark results.

Keywords. X-Ray, WebCL, Virtual Intraoperative Cholangiogram

1. Introduction

This paper describes the design and development of the Virtual Intraoperative Cholangiogram (VIC) training platform using WebCL. Laparoscopic cholecystectomy (LC) is one of the most commonly performed procedures worldwide [1]. Over 750,000 LC procedures are performed every year in the USA [2]. LC was derived from attempts to improve patient outcomes, decrease the length of stay at the hospital and minimize post-surgery morbidity. One of the major complications in LC is bile duct injury (BDI). Over the years, several strategies have been employed to minimize the incidence of BDI, including the use of intraoperative cholangiograms.

Cholangiogram is the imaging of bile duct using X-Rays. This imaging technique is used to visualize and examine the biliary tract. Cholangiogram is generated by injecting fluorescent fluids to the bile duct to create high contrast in the X-Ray image. The original purpose of the cholangiogram is to minimize BDI by clearly visualizing and correctly identifying the anatomy. However, a survey by Sanjay et al. [3] shows that only half of 20 surgical trainees and 20 consultant general surgeons could correctly identify normal anatomy. Only one third of the surgeons and surgeon trainees could correctly identify

[1] At the time of submission of the manuscript, the author was affiliated with University of Central Arkansas. The author is currently affiliated with University of Arkansas at Little Rock.

[2] Tansel Halic, Department of Computer Science, University of Central Arkansas, 201 Donaghey Avenue, Conway, AR, 72035, tanselh@uca.edu

the variations of normal anatomy in IC. It is evident in [3] that training the surgeons on biliary anatomy including various anatomical cases can increase the accuracy rate of IC interpretation [3].

In this study, we develop an accessible, portable, web-based VIC training platform. This platform allows users to visualize any 3D bile anatomy and generate realistic cholangiograms on any device. Our web-based training platform will help surgeons recognize variations of the anatomy. The immediate recognition of the different anatomy reduces BDI and thus decreases mortality and morbidity rate [4]. In our platform, cholangiograms are created with simulation of virtual X-Rays emitted from an X-Ray source to the detector. The attenuation of the X-Ray beam reflects gray color value in a final 2D X-Ray image. The simulation to create 2D IC image from 3D scene has $O(n^2m)$ time complexity; where nxn is the image size and m is the amount of polygons in the scene. This time complexity precludes interactivity especially for high resolution images (e.g. 2Kx2K). Therefore, we developed a parallel algorithm using WebCL, a JavaScript binding to OpenCL for heterogeneous parallel computing in the web browser [5], for fast generation of IC images as part of our training platform. We chose WebCL as a parallelization platform for VIC due to true cross-platform support and web accessibility without needing any special software/platform installed on the user's computer.

2. Design & Implementation

The first step is to import a 3D model into the scene. We used Pi-SoFMIS [6] to import 3D files in the scene. Once the scene is loaded, a material X-Ray attenuation constant (σ) is assigned to each model. The next step is to choose an output image size. The image size affects the number of the computational threads (kernels) launched and spatial partition of the target scene. Subsequent to the defining image size, each pixel on the output is mapped to an origin point in the scene. This origin point is the location of the X-Ray beam source. The next step is emitting X-Rays for every pixel originating from the source to the scene. During this ray casting, the penetration length needs to be computed as the X-Ray beam passes through a specimen or part of the specimen until it hits the detector. The X-Ray beam strength diminishes based on the physical properties of the material and penetration length. The penetration length is then used to calculate the grey value that is then mapped back to the image for producing the final X-Ray image.

2.1. Ray Cast & Ray Penetration

A ray per pixel is casted from the eye against the object. Möller–Trumbore [7] ray-triangle intersection algorithm is used as a method for a fast ray and triangle intersection in the scene. After finding all the intersection points, they are sorted based on distance from the X-Ray beam source to ensure that penetration length calculation is accurate when dealing with concave shapes and the penetration lengths of the intersecting rays are calculated and then sent to Grey Value (GV) function for further processing.

2.2. X- Ray Beam Attenuation

Through the use of Beer-Lambert Law [8], the transmittance (T) can be described as below;

$$T = e^{-\sum_{i=1}^{N} \sigma_i \int_0^l n_i(z)dz} \quad (1)$$

In Eq. (1) [6], σ_i is the attenuation cross section of the attenuating species *i* in the material sample, n_i is the number density of the attenuating species *i* in the material sample, *N* is the number of mass in the direction of the beam and ℓ is the penetration length of the beam of light through the material sample [9]. Since the mass is assumed to be homogenous, uniform attenuation is used and Eq. (1) can be reduced to:

$$T = e^{-\sum_{i=1}^{N} \sigma_i n_i l} \quad (2)$$

In Eq. (2), with the transmittance property calculated, the next step is to simulate the strength of the X-Ray beam. Since the X-ray beam fires photons, it is possible to calculate reduction in the number of photons passing through the masses on the path as given below;

$$P_{(E)} = P_{0(E)} \cdot e^{-\sum_{i=1}^{N} \mu_i l_i} \quad (3)$$

In Eq. (3), $P_0(E)$ is the number of X-Ray photons before the attenuation process at a specific energy spectrum and $P(E)$ is the number of X-Ray photons after the attenuation process at a specific energy spectrum.

$$GV = \sum_{E=E_1}^{E_{max}} P_{(E)} DE_{(E)} \quad (4)$$

In Eq. (4), $P(E)$ across all energy spectrums can be found and summed by repeating Eq. (3) with photon rays of different energy spectrums. *DE* refers to the detector efficiency coefficient at a specific energy spectrum. The sensitivity of the beam detector can be simulated by incorporating an additional coefficient that affects the intensity of the grey value [10]. The image is then generated when the grey value (GV) of each pixel is calculated.

2.3. Parallelization

Since each ray casting is a computationally independent task, they can be executed concurrently. The task is divided into spatial blocks based on the final image. Each block performs the computation on a specific part of the scene. Each pixel in these blocks is assigned to a single thread. A thread in the block is responsible for computing the *GV* of one single X-Ray. The number of threads in the block is equal to the total number of blocks. The initial coordinates of the X-Ray are calculated on block number and thread ID. The Pseudo code of the implementation is given in Table 2.

Table 2. Parallel Implementation Pseudo Code.

```
Divide image into blocks( X_BLOCKS * Y_BLOCKS )
Find geometry relevant to each block and store in geometryblock[x][y]
for each x in [0, X_DIMENSION]
for each y in [0, Y_DIMENSION]
      Allocate memory for geometryblock[x][y] on DEVICE
      Transfer data from HOST to DEVICE: σ, x, y, geometryblock[x][y]
      Execute kernel GVCalculator (σ, x, y, geometryblock) on DEVICE
      Read results from DEVICE: GV[x][y]
      Free memory
//Calculate penetration length and updates grey value (GV)
//Inputs: σ = attenuation constant, x= x index of geometryblock, y= y index of geometryblock,
geometryblock  = array of triangles relevant to the area
//Outputs: GV = array of final color value for a specific block
__kernel GVCalculator(σ, x, y, geometryblock  ) {
      Determine origin of the ray to cast based on x, y, and thread_id
      For each triangle in geometryblock
```

```
        Cast ray and save intersections
        Find all the penetration lengths based on intersections
        Calculate GV with thread_id, attenuation coefficient (σ) and penetration length
}
```

3. Results and Benchmarks

For our sample scene, a scene with biliary duct anatomy was created. The scene is easily modifiable as each object has its own material property constant. The region in biliary tract where the fluid is injected assigned to the material coefficient of the fluid. The platform is capable of rendering multiple X-Ray beams with distinct energy spectrums. Figure 1 shows the simulated VCI scene with different material properties and a single energy spectrum.

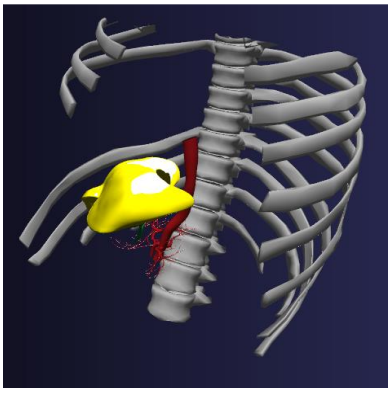

Figure 1a. Scene with different material properties represented by different colors

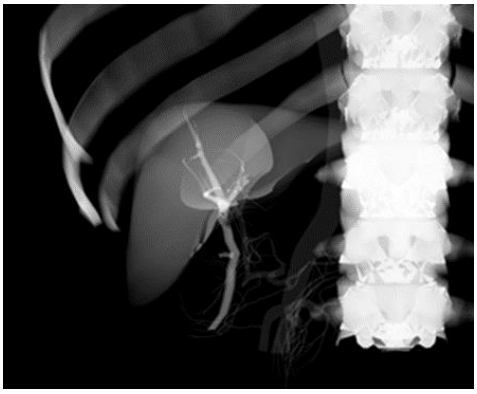

Figure 1b. Simulated VCI image generated from scene in Fig. 1a

In the benchmark, the scene was tested with varying parameters: the image size and the block size. The amount of triangles in the scene was kept constant at ~20,000. Box size is used to determine the amount of threads launched as well as the amount of local geometry that needs to be processed. The box sizes used in the scene were 32x32, 64x64, 128x128, and 256x256. The used image sizes for testing were 128x128, 256x256, 512x512, 1024x1024, 2048x2048, and 4096x4096. Each different image size was rendered using four different box sizes given above e.g. 32x32.

Benchmark data was collected on Mozilla Firefox Version 32.0 using the Nokia Add-On to reveal JavaScript to OpenCL. The specifications of the computer used were; 64-bit Intel i7-3770 CPU with 3.40 GHZ, a 16 GB memory and a GeForce GT640 Driver version 347.25 with Windows 7.

In order to assess speed-up, we developed our algorithm using serial implementation of JavaScript, native execution (C code) and OpenMP. The graphs in Figure 2 show these benchmarks for JavaScript (Serial), C (Serial), C (OpenMP), WebCL (GPU) and WebCL (CPU) for each box size with varying image resolution. In our results, computation time required to calculate an entire VCI scene significantly decreases as the box size and image size increases. When the box size is small, the speed-up is limited since data needs to be constantly read from host and written on the device memory. Figure 3 shows the speedup comparison for C (Serial), C (OpenMP), WebCL (GPU),

and WebCL (CPU) against JavaScript serial version. By examining this graph, all implementations are significantly faster than JavaScript. WebCL speeds are capable of exceeding compiled parallel C code in cases where the block size is greater than 128 and when the image is resolution is higher than 1024x1024.

Figure 2. Time required for each box size

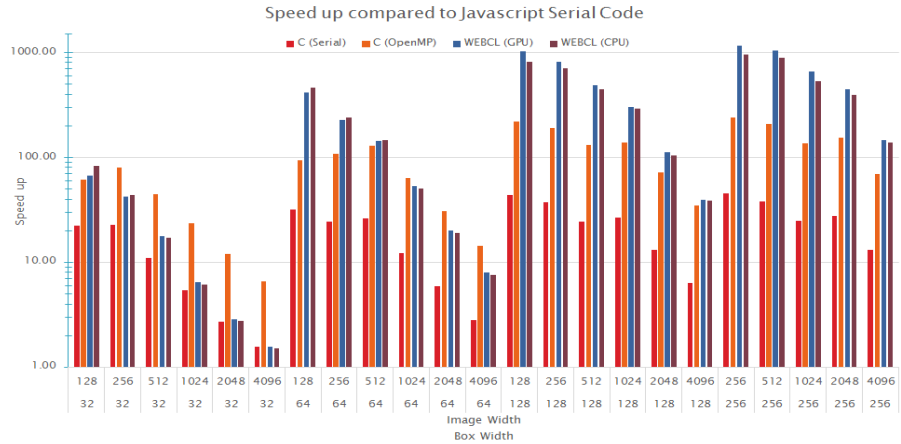

Figure 3. Speedup comparisons with respect to the serial execution in the web browser

4. Conclusion and Future Works

In this study, we introduced a web based VIC training platform. In our platform, we developed parallel WebCL based IC algorithm and analyzed its performance. The benchmarks of the algorithm include comparison of JavaScript, C serial, C OpenMP, WebCL (CPU) and WebCL (GPU) implementations. Based on our results, the OpenMP and WebCL attained the highest execution speed-ups. We were able to exceed execution

speeds of compiled OpenMP code with the use WebCL with the block size greater than 128 and the image resolution higher than 1024x1024. In the future, we would like to extend our study to integrate anisotropic material properties with material mapping to increase the realism and physical accuracy.

Acknowledgments

This publication was made possible by the Arkansas INBRE program, supported by grant funding from the National Institutes of Health (NIH) National Institute of General Medical Sciences (NIGMS) (P20 GM103429) (formerly P20RR016460).

References

[1] E. Chekan, M. Moore, T. D. Hunter, and C. Gunnarsson, "Costs and Clinical Outcomes of Conventional Single Port and Micro-laparoscopic Cholecystectomy," JSLS, vol. 17, no. 1, pp. 30–45, 2013.

[2] C. M. Vollmer and M. P. Callery, "Biliary injury following laparoscopic cholecystectomy: why still a problem?" Gastroenterology, vol. 133, no. 3, pp. 1039–1041, Sep. 2007.

[3] P. Sanjay, S. Tagolao, I. Dirkzwager, and A. Bartlett, "A survey of the accuracy of interpretation of intraoperative cholangiograms," HPB, vol. 14, no. 10, pp. 673–676, Oct. 2012.

[4] F. R. Polat, I. Abci, I. Coskun, and S. Uranues, "The Importance of Intraoperative Cholangiography during Laparoscopic Cholecystectomy," JSLS, vol. 4, no. 2, pp. 103–107, 2000.

[5] "WebCL," *WebCL*. [Online]. Available: http://webcl.nokiaresearch.com/.

[6] T. Halic, W. Ahn, and S. De, "A framework for web browser-based medical simulation using WebGL.," in *MMVR*, 2012, pp. 149–155.

[7]T. Möller and B. Trumbore, "Fast, Minimum Storage Ray-triangle Intersection," J Graph Tools, vol. 2, no. 1, pp. 21–28, Oct. 1997.

[8] D. F. Swinehart, "The beer-lambert law," *J. Chem. Educ.*, vol. 39, no. 7, p. 333, 1962.

[9] F. P. Vidal, M. Garnier, N. Freud, J. M. Létang, and N. W. John, "Simulation of X-ray attenuation on the GPU," in *Theory and Practice of Computer Graphics*, 2009, pp. 25–32.

[10] M. Reiter, M. M. Malik, C. Heinzl, D. Salaberger, E. Gröller, H. Lettenbauer, and J. Kastner, "Improvement of X-Ray image acquisition using a GPU based 3DCT simulation tool," in *8th International Conference on Quality Control by Artificial Vision*, 2009, p. 8.