平成 23 年 9 月 30 日

GPGPU を用いた仮想窓システムの高速化

電気通信大学 情報工学科 成見研究室
0811088 春田 英和

1 はじめに

近年、3D テレビなどの 3D 立体視が身近になっている。しかし 3D を生かしたコンテンツは決して多くはない。また、3D がより効果的な高精細な表示装置を使うとなるとその数は必然的に少なくなる。そのような背景から、高精細な 3D コンテンツとして「仮想窓」システム [1] が作られた。仮想窓システムとはディスプレイの前で観察者が動くと、観察者の視点の位置に合わせ最適な画像を表示し、まるでそこに本当の窓があるかのように見せるシステムである。

しかし昨年度開発されたシステムは動作が遅く、コンテンツとしては成り立たない。本研究ではこの仮想窓システムの画像切り替え部分を GPGPU を用いて高速化することを目標とする。

2 仮想窓システム

仮想窓システムは高精細に撮影された風景の多視点の画像を用意し、そこから観察者の見る位置によって窓と見立てたディスプレイに表示する画像を変化させるシステムである。また、より立体感を得るため 3D 立体視と巨大な表示装置を用いる。多視点の立体画像を収集し表示することで人が右に動けば物体の右側を見ることができる。

2.1 自動撮影システム

仮想窓システムでは少しずつカメラをずらしながら 2 次元配列状に大量の画像を撮影する必要がある。その撮影作業を全て手作業で行うと膨大な時間がかかる上に精度も落ちる。そのため小刻みに動いては撮影を行う装置がある。台座の移動には LEGO Mindstorm、撮影にはリモートコントロールソフトの Canon RemoteCaptureDC を用いている。

自動撮影した画像のうち、対象としている背景画像は以下のものである。

- 合計数：横 101 枚×縦 26 枚＝ 2626 枚
- 解像度：3456 × 2592
- ファイルサイズ：一枚約 800KB × 2626 枚＝合計約 2.1GB

2.2 画像切替システム

画像の切り替えには観察者の視点位置情報を取得し、最適な画像を選出し、読み込み、テクスチャ化した画像を表示するという流れがある。

2.2.1 観察者の視点位置座標の取得

視点位置検出には Web カメラと ARToolKit の AR マーカーを用いている。仮想窓システムでは ARToolKit の物体認識部分を用いている。Web カメラが AR マーカーを認識すると、カメラとの相対的な 3 次元座標が求められる。AR マーカーを 3D メガネに取り付け、Web カメラをディスプレイ中央上部に取り付ける。

2.2.2 最適な画像の選出・読み込み

自動撮影装置より一定間隔で撮影した 2 次元配列状の素材画像から立体視のため 2 枚の画像を選出する。選出された画像を OpenCV ライブラリを用いて読み込み、OpenGL でテクスチャ化する。

2.3 各処理における実行時間

表 1 に各処理にかかる実行時間を示す。

表 1　実行時間

処理	実行時間 (秒)
マーカー認識	0.04
適切な画像の選択	0.001
画像の読み込み	0.5
テクスチャ化	0.05
表示	0.004

3 達成目標

本研究では 2.3 に示した実行時間の内、画像の読み込みを高速化することを達成目標とする。具体的な数字として、1 秒間に 30 回以上の表示切替を目標とすると、全ての処理が 0.033…秒以下である必要がある。「画像の読み込み」「切り替え」のみ高速化しただけでこれを達成するのは不可能であるが、現在最も処理が重い部分であることから「画像の読み込み」「切り替え」の処理における実行時間を 0.03 秒以下にすることを当面の目標とした。これを実現するために GPGPU の技術を使用する。現在はハードディスク上のファイルを OpenCV で読み込んで、OpenCV を用いて表示しているが、これらの処理を全て GPU 上に搭載することを目指す。

4 これまでの研究

4.1 動画の作成

実行時間の短縮をするため、まず画像の読み込みを全てメモリからできるようにする必要がある。最終的に GPU で画像処理することを視野に入れると GPU のメモリが 1GB 程度なので容量の合計を 1GB 以下にする必要がある。そこで解像度を落とさずデータ量を落とす為に画像を動画形式に圧縮する方法をとる。圧縮した

動画ファイルをメモリ上に保持し、その動画から任意のフレームを取得し、表示に使うという形をとる。また、CUDA のデコーダーに対応し、高い圧縮率を期待できる H.264 形式の動画を作成する。

動画作成にあたりまず TMPGEnc でビットレート 50Mbps、フレームレート 24fps、12 フレーム毎にキーフレームという設定で連番画像を mpeg1 形式の動画にした。この動画を H.264 形式の動画にする。以下のようにいくつかのフリーソフトを試した結果 BatchDoo!というソフトが最も性能がよかった。

表 2　エンコードソフト比較

	解像度	ファイルサイズ	備考
ソース画像	3456 × 2592	約 80MB	横一列 (101 枚)
BatchDoo!	3456 × 2592	約 8MB	
Xmedia Recode	2048 × 1536	約 8MB	コマ落ちが発生する
Media Coder			エンコードできない
TMPGEnc ver.5	解像度設定に限界がある		

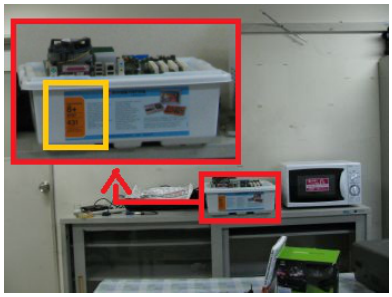

図 1　比較箇所

図 2　画質比較

ソース画像、mpeg1 動画、Xmedia Recode でエンコードした動画、BatchDoo!でエンコードした動画のキャプチャ画像で画質を比較する。図 1 の赤で囲んだ箇所の内、黄色の文字が書いてある箇所を比較した図が図 2 である。BatchDoo!でエンコードした動画はファイルサイズ、解像度、画質とも良好であった。

4.2　OpenCV による任意フレーム取得 [2]

作成した動画を OpenCV でデコードする。OpenCV の関数を用いて動画の単純な再生はできた。実際に用意した動画で再生したところ、およそ 10fps で再生することができた。つまり 1 枚につき 0.1 秒のデコード時間が必要である。しかし任意のフレームだけデコードする場合には更に問題がある。動画のプロパティ情報に値を設定することでフレーム単位での位置を設定できるが実際はキーフレーム毎にしか任意のフレーム移動はできない。用意した動画は 12 フレーム毎にキーフレームがあるので最小では 0.1 秒のデコードで済むが最大で約 1 秒のデコードを要してしまう。これでは 0.33 秒以下での処理は難しい。

5　CUDAvideodecoderAPI[3] による任意フレーム取得

GPU で任意フレームの取得をするために CUDA の動画デコード API を使用した。まずこの API を使って動画情報の取得と動画の再生をするサンプルプログラムを用いてテストを行った。サンプルプログラムに付属する動画は正しく動作するが、今回作成した H.264 動画では動作しなかった。フレームのサイズやビットレートなどの動画情報が正しく認識されていないようであった。結局、用意した動画ではプログラムの途中でエラーが起き、動画が再生されることはなかった。

6　まとめと今後の課題

仮想窓システムで最も時間のかかる画像の読み込み・切り替えの処理の高速化を目指した。読み込みの高速化のために画像を動画として圧縮することで、今までの 10 分の 1 にファイルサイズを小さくできた。しかし動画の任意のフレームをデコードするために、OpenCV では時間がかかりすぎ、CUDA 動画デコード API では動作しなかった。

今後はまず CUDA のサンプルプログラムを改良し、GPU を用いた任意フレームの取得ができるプログラムを作る。また、そのプログラムで用意した動画のフレームが正常に取得できない場合、新しく動画を準備することを考えている。

参考文献

[1] 清水陽介 『大型高精細ディスプレイを用いた「仮想窓システム」』電気通信大学電気通信学部情報工学科成見研究室平成 22 年度卒業研究

[2] 画像処理ソリューション http://imagingsolution.blog107.fc2.com/blog-entry-210.html

[3] NVIDIA CUDA VIDEO DECODER http://developer.nvidia.com/cuda-toolkit-32-downloads よりダウンロードした CUDA SDK 内の cudaDecodeGLdocnvcuvid.pdf