# A GPU Implementation of Level Set Multiview Stereo

Patrick Labatut[1], Renaud Keriven[2], and Jean-Philippe Pons[2]

[1] Département d'Informatique, École normale supérieure,
F-75230 Paris Cedex 05, France,
patrick.labatut@ens.fr

[2] CERTIS, École Nationale des Ponts et Chaussées,
F-77455 Marne-la-Vallée Cedex 2, France,
keriven@certis.enpc.fr, pons@certis.enpc.fr

**Abstract.** Variational methods that evolve surfaces according to PDEs have been quite successful for solving the multiview stereo shape reconstruction problem since [1]. However just like every other algorithm that tackles this problem, their running time is quite high (from dozens of minutes to several hours). Fortunately graphics hardware has shown a great potential for speeding up many low-level computer vision tasks. In this paper, we present the analysis of the different bottlenecks of the original implementation of [2] and show how to efficently port it to GPUs using well-known GPGPU techniques. We finally present some results and discuss the improvements.

## 1 Introduction

Three-dimensional shape reconstruction from a set of pictures is one of the oldest problems in computer vision and find its roots back in robotics. Unfortunately the current state-of-the-art algorithms for reconstruction from multiple views are typically very slow and forbid a more widespread use of this technique.

A quite recent idea to improve the running time of many computer vision algorithms consists in using commodity graphics cards not for rendering fancy graphics but for general-purpose computations. We show how this approach was successful for us, allowing quality shape reconstruction within minutes.

The first section of this paper discusses previous work in the field of stereo reconstruction and general-purpose computation on GPUs (GPGPU), the next section describes the shape reconstruction algorithm we used, the following section details our implementation and the final section presents some results.

## 2 Related Work

### 2.1 Multiview Stereo Algorithms

Given $n$ ($\geq 2$) images of the same scene (along with the calibration parameters of the cameras), the goal is to build a 3D model of the scene as close as possible

to the original. This goal is difficult to reach because occluded parts and lighting can substantially change the appearance of a scene from different viewpoints.

Currently multiview stereo algorithms can be very roughly divided into two classes: on one hand, discrete methods à la *space carving* derivated from [3] which work on an initialy whole discrete volume and incrementally remove chunks of voxels that do not satisfy a photo-consistency condition; on the other hand, variational methods based on the deformation of a surface under a PDE [1, 4]. The algorithm considered here [2] belongs to this latter class.

### 2.2 General-Purpose Computation on GPUs

In just a few years graphics cards have become heavily parallel processing machines with increased programming capabilities making their use possible for other purposes than standard real-time rendering [5]. A simplistic way to understand what a GPU actually does is to consider it as a stream processor [6] which executes a computational kernel over all the elements of an input stream (possibly accessing other streams) and puts the corresponding results into an output stream.

### 2.3 Stereo Vision Using Graphics Hardware

Computer vision algorithms are nowadays often GPU-accelerated, as they work on the same kind of data as rendering. Recovering the disparity map of two images has already been thoroughly studied: from simple block matching strategy with a multiscale approach [7], to mixed CPU/GPU approach initializing a graph-cut optimization with crude depth maps computed on GPU [8], and parallel dynamic programming on GPU [9]. Numerical schemes for 2D level sets have also been implemented by brute force [10] and more recently, 3D level sets for segmentation [11] introduced a GPU to CPU message passing system. However, to our knowledge, no multiview stereo algorithm for full shape reconstruction has been adapted to run on GPUs.

## 3 Shape Reconstruction Method

The variational method of [2] borrows from [1] but formulates the evolution of the surface as an image registration problem. It is thus simpler, more robust than most other methods and also more suitable for a GPU implementation.

### 3.1 Notations

A surface $S \subset \mathbb{R}^3$ models the shape of the scene being reconstructed. We note $P_i : \mathbb{P}^3 \to \mathbb{P}^2$ the projection matrices and $I_i : \Omega_i \subset \mathbb{R}^2 \to \mathbb{R}^c$ the corresponding images. $S_i$ is the part of the surface $S$ visible in the image $I_i$. $P_{i,S}^{-1} : P_i(S) \to S_i$ reprojects from the camera $P_i$ to the surface $S$. Finally $I_j \circ P_j \circ P_{i,S}^{-1} : P_i(S_j) \to \mathbb{R}^c$ is the reprojection of the image $I_j$ in the camera $P_i$ via the surface $S$.

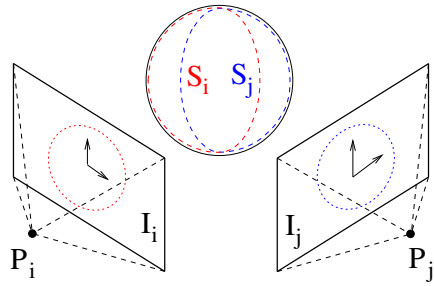



**Fig. 1.** Cameras setup and notations

### 3.2 Energy to Minimize

We wish to minimize a sum of dissimilarity terms between pairs of images: each pair is composed of one of the input images and some predicted image obtained by reprojecting another input image into the camera corresponding to the former image. This leads to the following energy ($M$ is the dissimilarity measure between two areas of image):

$$\mathcal{M}(S) = \sum_i \sum_{j \neq i} \mathcal{M}(I_i, I_j)(S) = \sum_i \sum_{j \neq i} M|_{\Omega_i \cap P_i(S_j)} (I_i, I_j \circ P_j \circ P_{i,S}^{-1}) \tag{1}$$

The minimization of this energy results in the evolution of the surface $S$ along its outward normal $\mathbf{N}$, driven by the equation:

$$\frac{\partial S}{\partial t} = \left[ -\lambda H + \sum_i \sum_{j \neq i} \delta_{S_i \cap S_j} \, \partial_2 M \, DI_j \, DP_j \, \frac{\mathbf{d}_i}{z_i^3} \right] \mathbf{N} \tag{2}$$

where $H$ is the mean curvature of $S$ (which corresponds to a smoothing term added to the energy), $D\cdot$ is the Jacobian matrix of a function, $\delta_{\cdot}$ is the Kronecker symbol, $\mathbf{d}_i$ the vector from the camera $P_i$ to the considered point, $z_i$ its depth and $\lambda$ a smoothing coefficient ($\lambda > 0$).

### 3.3 Similarity Measure

The described method allows the use of whatever similarity measure we want: cross-correlation, correlation ratio, mutual information, etc... [12]. We limited ourselves to the *local normalized cross-correlation* $\mathrm{cc}(I_i, I_j)$ (which can accomodate even non-lambertian surfaces provided the window size is small enough):

$$\begin{aligned} \mu(I_i) &= \tfrac{G_\sigma * I_i}{\omega} & \mathrm{v}(I_i) &= \tfrac{G_\sigma * I_i^2}{\omega} - \mu^2(I_i) + \tau^2 \\ \mathrm{v}(I_i, I_j) &= \tfrac{G_\sigma * I_i I_j}{\omega} - \mu^2(I_i)\,\mu^2(I_j) & \mathrm{cc}(I_i, I_j) &= \tfrac{\mathrm{v}(I_i, I_j)}{\sqrt{\mathrm{v}(I_i)\,\mathrm{v}(I_j)}} \end{aligned} \tag{3}$$

where $\omega(\mathbf{x_0}) = \int_\Omega G_\sigma(\mathbf{x_0} - \mathbf{x})\, d\mathbf{x}$ is the spatial normalization to account for the shape of the correlation window, and $G_\sigma$ is a gaussian kernel.

The dissimilarity measure $M^{cc}(I_i, I_j)$ between images $I_i$ and $I_j$ is simply the sum of the normalized cross-correlation over the whole domain: $-\int_\Omega \mathrm{cc}(I_i, I_j)(\mathbf{x})\, d\mathbf{x}$ and its partial derivative needed for the minimization is:

$$\partial_2 M^{\mathrm{cc}}(I_i, I_j) = \alpha(I_i, I_j)\, I_i + \beta(I_i, I_j)\, I_j + \gamma(I_i, I_j) \tag{4}$$

where:

$$\begin{aligned} \alpha(I_i, I_j) &= G_\sigma * \frac{-1}{\omega \sqrt{\mathrm{v}(I_i)\, \mathrm{v}(I_j)}} \\ \beta(I_i, I_j) &= G_\sigma * \frac{\mathrm{cc}(I_i, I_j)}{\omega\, \mathrm{v}(I_j)} \\ \gamma(I_i, I_j) &= G_\sigma * \left( \frac{\mu(I_i)}{\omega \sqrt{\mathrm{v}(I_i)\, \mathrm{v}(I_j)}} - \frac{\mu(I_j)\, \mathrm{cc}(I_i, I_j)}{\omega\, \mathrm{v}(I_j)} \right) \end{aligned} \tag{5}$$

### 3.4 Energy Minimization

The minimization of the energy by gradient descent is implemented within the *level set* framework [13] and can implicitly cope with surface topology changes. However this comes at a cost and to reduce the computational burden, the *narrow band* algorithm [14] is used to evolve the level sets. As the energy is optimized through a simple steepest gradient descent, it can easily get stuck in a local minimum. The algorithm therefore adopts a multi-scale approach by using the result of the optimization at a coarser scale to initialize the optimization at a finer level.

## 4 Graphics Hardware Implementation

Whereas other variational methods for multiview stereo are CPU-only, [2] was designed with classical GPU acceleration in mind. We take this one step further by using GPGPU techniques.

### 4.1 Original Implementation Analysis

The main loop driving the evolution of the surface and executed at each time step can be decomposed as shown in Tab. 1. As all the surface points visible in image $I_i$ should be points from the narrow band, the $M^{\mathrm{cc}}$ derivative is computed over the common domain of image $I_i$ and image $I_j$ reprojection, allowing for stream computation. Items 6 and 7 actually spend most of the time doing bilinear interpolations. The depth computations and reprojections were already running on GPU. Finally the level sets computations cover only a fraction of the running time. We thus chose to concentrate our efforts on items 5.2, 6 and 7.

| | |
|---|---|
| 0 % | 1 mesh update |
| 10 % | 2 distance function update |
| 5 % | 3 mesh download to the GPU |
| 5 % | 4 depth computation in every camera<br>5 similarity measure derivative update<br>for each camera couple $(i, j)$ |
| 0 % | 5.1 reprojection of the image $I_j$ in the camera $P_i$ |
| 20 % | 5.2 computation of the similarity measure derivative |
| 20 % | 6 computation of band points attributes<br>for each band point / for each camera<br>position / visibility / intensity computation |
| 40 % | 7 normal speed computation<br>for each band point / for each camera pair<br>if the point is visible in the two cameras<br>7.1 corresponding normal speed computation |
| 0 % | 8 CFL condition |
| 0 % | 9 level sets update |

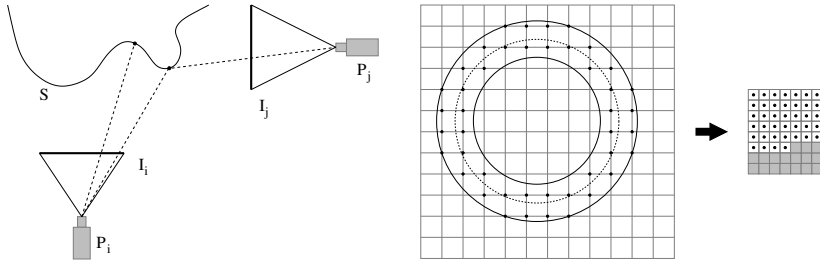
**Table 1.** Main loop with typical running time distribution

### 4.2 Reprojection and Visibility Masks Computation

The depth is computed via conventional rendering of the surface and update of the *depth buffer*. The visibility masks $(\Omega_i \cap P_i(S_j))$ and the image reprojections are computed with the *shadow mapping* technique which consists in using the contents of the *depth buffer* we got from the $P_j$ camera as a texture and rendering the surface in the camera $P_i$. Accessing texels in this special texture triggers a comparison between the current depth and the depth stored in the texture and returns a boolean value. The surface points are used as texture coordinates and the texture matrix (which is applied to the texture coordinates before accessing *texels*) is replaced by the $P_j$ camera matrix. We can thus generate a depth mask using the $P_j$ camera *depth buffer* as a texture. Then the $I_j$ image reprojection is obtained by applying this image as a texture (see Fig. 2(a)).

### 4.3 Computation of the Similarity Measure Derivative

The convolutions were originally implemented with a recursive filter [15]. IIR filters do not fit very well in the GPU computational model constraints so it was replaced by a simple separable convolution. In order to mask away some of the input records in the stream, we take advantage of the efficient *Z-Culling* technique: it consists in loading a mask in the *depth buffer* that allows masked records to completely skip the execution of the computational kernel. Using this masking technique, the computation of $\alpha$, $\beta$ and $\gamma$ from (5) easily maps to successive kernels: first compute 1, $I_i$, $I_j$, $I_i\, I_i$, $I_i\, I_j$, $I_j\, I_j$, then convolve the previous pass result with $G_\sigma$, then compute $\omega, \mu(I_i)$, $\mu(I_i)$, $\mathrm{v}(I_i)$, $\mathrm{v}(I_j)$, $\mathrm{v}(I_i, I_j), \mathrm{cc}(I_i, I_j)$ and finally convolve with $G_\sigma$ to get $\alpha$, $\beta$ and $\gamma$.




(a) shadow mapping (b) mapping the band to a 2D stream

**Fig. 2.** Some of the techniques used

### 4.4 Computation of the Points Position, Visibility, Intensity and Normal Speed

At the finest scale, the band typically contains many dozens of thousand of points. An input stream containing the coordinates of the band points is first created (as shown in Fig. 2(b)). Computational kernels iterate over the cameras, compute the position, visibility and intensity attributes from this input stream, and output corresponding attributes streams. *Z-Culling* is once again used to mask away some parts of the input stream where no computation needs to be done.

For the normal speed we combine visibility streams and the stream mask to generate a mask for *Z-Culling*. The camera pairs are then iterated over while accumulating the normal speeds computed for each points in the narrow band. The level sets are finally updated using this output stream.

## 5 Results

The graphics hardware was programmed using the OpenGL API and its extensions mechanism. The Cg programming language was also used for prototyping. All the presented results (Tab. 2 and Fig. 3) were obtained on a PC with an Intel Xeon 2.8 GHz CPU and 1 GB of system RAM equipped with an NVIDIA GeForce 7800 GTX graphics card with 256 MB of video RAM.

The overall speed factor is almost about 4 when compared to the already GPU accelerated method from [2]. However the original sections of the algorithms that were using lots of bilinear interpolations observe an elevenfold improvement in general, and the computation of the measure derivative gets a ninefold decrease of its running time.

**Table 2.** Input data, parameters and running times

| | "buddha"[1] | "dino"[2] |
|---|---|---|
| #Images | 25 | 16 |
| Resolution | $256 \times 256 \times 3$ | $256 \times 256 \times 3$ |
| #Pairs | 50 | 32 |
| Level set volume | $128 \times 128 \times 128$ | $192 \times 192 \times 192$ |
| Running time (CPU/GPU) | $\sim 780$ s | $\sim 860$ s |
| Running time (GPU) | $\sim 210$ s | $\sim 240$ s |

## 6 Conclusion

The decrease of the total running time is significative allowing shape reconstruction within minutes and it is even more impressive if we consider the CPU-only method from [1]: the hypothetical overall performance gain would be about 200.

## References


[1] Faugeras, O., Keriven, R.: Complete dense stereovision using level set methods. In: European Conference on Computer Vision. Volume 1406. (1998) 379–393

[2] Pons, J.P., Keriven, R., Faugeras, O.: Modelling Dynamic Scenes by Registering Multi-View Image Sequences. International Conference on Computer Vision and Pattern Recognition **2** (2005) 822–827

[3] Kutulakos, K., Seitz, S.: A Theory of Shape by Space Carving. International Journal of Computer Vision **38**(3) (2000) 199–218

[4] Duan, Y., Yang, L., Qin, H., Samaras, D.: Shape Reconstruction from 3D and 2D Data Using PDE-based Deformable Surfaces. In: European Conference on Computer Vision. Volume 3. (2004) 238–251

[5] Owens, J.D., Luebke, D., Govindaraju, N., Harris, M., Krüger, J., Lefohn, A.E., Purcell, T.J.: A Survey of General-Purpose Computation on Graphics Hardware. In: Eurographics 2005, State of the Art Reports. (2005) 21–51

[6] Buck, I., Foley, T., Horn, D., Sugerman, J., Fatahalian, K., Houston, M., Hanrahan, P.: Brook for GPUs: Stream Computing on Graphics Hardware. ACM Transactions on Graphics **23**(3) (2004) 777–786

[7] Yang, R., Pollefeys, M.: Multi-Resolution Real-Time Stereo on Commodity Graphics Hardware. In: IEEE Conference on Computer Vision and Pattern Recognition. Volume 1. (2003) 211–218

[8] Geys, I., Koninckx, T.P., Gool, L.J.V.: Fast Interpolated Cameras by Combining a GPU based Plane Sweep with a Max-Flow Regularisation Algorithm. In: International Symposium on 3D Data Processing, Visualization and Transmission. (2004) 534–541


[1] Intel OpenLF Mapping project: http://www.intel.com/research/mrl/research/lfm/

[2] Multiview Stereo Evaluation project: http://grail.cs.washington.edu/projects/mview/

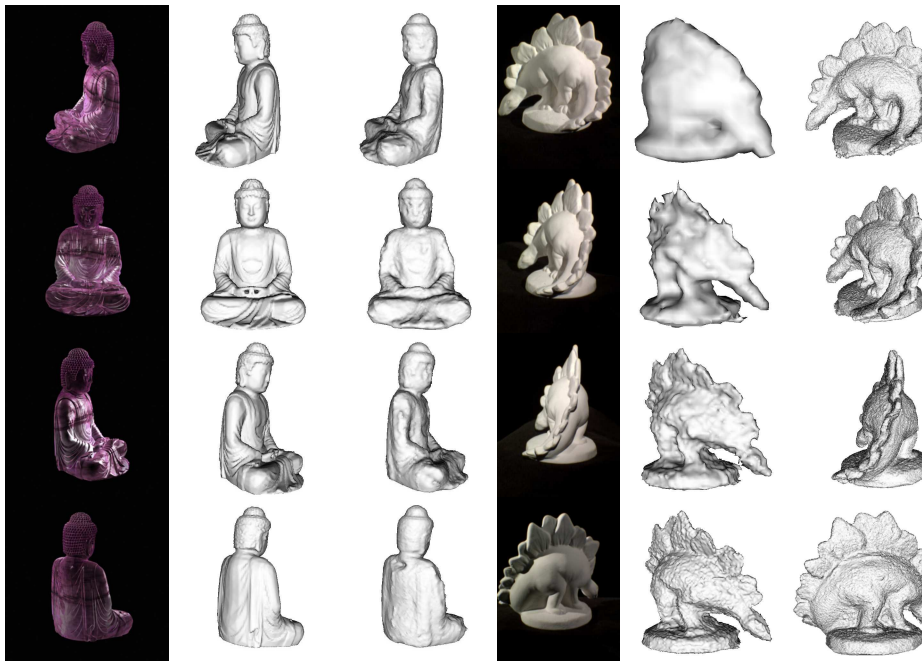


**Fig. 3.** "buddha" data set: first column: some input images, second column: ground truth, third column: result; "dino" data set: first column: some input images, second column: evolution (first time step of each scale), third column: result


[9] Gong, M., Yang, Y.H.: Near Real-Time Reliable Stereo Matching Using Programmable Graphics Hardware. In: IEEE International Conference on Computer Vision and Pattern Recognition. (2005)

[10] Rumpf, M., Strzodka, R.: Level Set Segmentation in Graphics Hardware. In: IEEE International Conference on Image Processing. Volume 3. (2001) 1103–1106

[11] Lefohn, A.E., Kniss, J.M., Hansen, C.D., Whitaker, R.T.: A Streaming Narrow-Band Algorithm: Interactive Deformation and Visualization of Level Sets. IEEE Transactions on Visualization and Computer Graphics **10**(40) (2004) 422–433

[12] Hermosillo, G., Chefd'hotel, C., Faugeras, O.: Variational Methods for Multimodal Image Matching. International Journal of Computer Vision **50**(3) (2002) 329–343

[13] Osher, S., Sethian, J.A.: Fronts Propagating with Curvature-Dependent Speed: Algorithms Based on Hamilton-Jacobi Formulations. Journal of Computational Physics **79**(1) (1988) 12–49

[14] Adalsteinsson, D., Sethian, J.A.: A Fast Level Set Method for Propagating Interfaces. Journal of Computational Physics **118**(2) (1995) 269–277

[15] Deriche, R.: Fast Algorithms for Low-Level Vision. IEEE Transactions on Pattern Analysis and Machine Intelligence **1**(12) (1990) 78–88