Real-time multi-stereo depth estimation on GPU with approximative discontinuity handling

J. Woetzel and R. Koch
Institute of Computer Science and Applied Mathematics,
Christian-Albrechts-University of Kiel,
24098 Kiel, Germany
{jw, rk}@mip.informatik.uni-kiel.de

Abstract

This paper describes a system for dense depth estimation for multiple images in real-time. The algorithm runs almost entirely on standard graphics hardware, leaving the main CPU free for other tasks as image capture, compression and storage during scene capture.

We follow a plain-sweep approach extended by truncated SSD scores, shiftable windows and best camera selection.

We do not need specialized hardware and exploit the computational power of freely programmable PC graphics hardware. Dense depth maps are computed with up to 20 fps.

Keywords:
Real-time multi view stereo, dense depth estimation, programmable general purpose GPU, previsualization.

1 Introduction

Dense stereo reconstruction has been an active field of research for the last decades but still proposes challenging research problems. In particular the wide range of real-time applications require highest possible quality with the additional constraint of a hard restriction on the maximum processing time.

The wide range of applications range from navigation and geometrical 3D modeling to depth compensated image based rendering techniques. Capturing a highly complex scene may fail due to lighting situation or missing details in the chosen camera path. Our answer to this problem is live multi-stereo depth-estimation with real-time monitoring of scene capture and reconstruction. This allows the user to see a 3D reconstruction on the fly. Live-results are sufficient for pre-visualization and pre-production planning. The final high-end cinema production may require higher quality. Offline post-processing can be used to refine the real-time results with a robust fusion approach or a standard offline reconstruction method.

Typical problems of stereo algorithms include suitable handling of image noise, texture-less regions, depth discontinuities and occlusions. Additional constraints arise in the multiple view case of more than two cameras. For example points may be occluded in some cameras only which can be used to overcome occlusions. Non-overlapping field of view can be considered as border effect for the two view case. But in the multi-view case this has to be handled explicitly for general camera configurations.

Availability of truly free programmable graphics hardware has opened a very active field of research motivated by the enormous processing power of graphics cards. A simple comparison of the transistor count of current CPU and GPU raises high performance expectations with current graphic cards processors. For example a Pentium 4 processor has approx. 55 million transistors in comparison to approx. 125 million transistors of Nvidias Geforce FX graphics cards[6, 10]. A quantitative comparison [11] of sample implementations optimized for GPU and CPU confirms the processing power advantage of GPU against CPU for special situations but also detects bottlenecks like data transfer from GPU to CPU. To conclude, with standard PC's you get a specialized high-performance DSP almost for free which we will use in our approach to keep CPU free for other tasks.

This document is structured as follows. We start with an overview of related work. Then we will introduce the used hardware architecture followed by a description of our method. Results demonstrate the suitability of our approach before concluding remarks.

2 Related Work

Dense stereo algorithm have been explored for decades. A complete state of the art description is beyond the scope of this paper. We refer to a recent comparison by Scharstein et. al. [13] for a general overview.

Recent formulations as a global optimization problem by scan-line Dynamic Programming [5] and its 2D generalization by Graph Cut Minimization [9] obtain good results. But they are too slow for real-time applications because of their very complex optimization scheme, even for approximative iterative solutions. However, they are perfectly suited for a post-processing offline refinement of our real-time results similar to the approach presented by Kang [7].

To achieve real-time performance with software on CPU single instruction multiple data (SIMD) instructions like MMX and SSE were used. SIMD instruction sets allow parallel execution of lower precision data embedded into higher precision registers. A 32 bit register may be used to add four eight bit number simultaneously, for example. Fastest available CPU implementations are based on this approach [4, 12].

Restrictions of the search range and resolution affect performance directly. Assumptions on the scene can be used to minimize search space for example with pyramidal schemes [3].

In our approach we use the GPU efficiently to compute, score and select the best depth hypotheses for each pixel. Two algorithms exist that can deal with real-time constraints implemented in current graphics hardware [14, 15] and one with near real-time performance [16, 17]

Our approach is closest to Yangs approach [15] of multi-camera depth estimation. He is using a fixed five camera configuration and multi-pass rendering for estimation. He restricted his approach to two-view geometry and combined it with an aggregation of the dissimilarity measure by summation over a box-filtered LOD pyramid [14]. This can be computed efficiently in hardware by automipmap generation, but not for free. Please note that his first approach [15] is very sensitive to noise due to his simple winner-take-all (WTA) optimization with a small 1×1 support region. His second approach [14] can neither handle depth discontinuities correctly nor use more than two cameras.

The near-real-time approach of Zach et al. [16, 17] follows a different idea of iteratively refining a scene representation by an approximative surface triangulation. A regular grid for the point of view of a reference camera is refined based on local dissimilarity of stereo gray images. Their approach uses a complex iteration scheme at the cost of performance.

We follow the basic idea of a plane sweep algorithm [2] and extended it to

- an arbitrary configuration of currently up to eight cameras,
- dissimilarity measure truncation depending on signal to noise ratio (SNR) to limit mismatch and noise influence,
- a shiftable windows implementation,
- handling partial field of view overlap of multiple cameras,
- and a 'best n of m' and best half-sequence multi-camera selection to handle occlusions partly correct.

In addition we extended the simple plane sweep to a generalized sweep which can handle irregular highly complex search spaces of arbitrary geometry without loss of performance w.r.t hypothesis evaluations per second.

3 Hardware Architecture

It is important to understand the concepts of current GPU architecture to develop and adapt general purpose algorithms for implementation on GPU hardware. First of all, the GPU supports

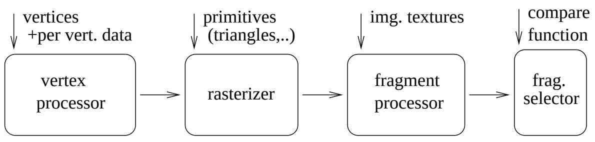

Figure 1: Simplified flow of data through GPU architecture pipeline from vertex processor via rasterizer to fragment processor. One or no fragment is selected for final rendered pixel output.

vector operations (SIMD). Currently four component vectors are natively supported without performance loss. Secondly the architecture is pipelined with a very restricted flow of data compared to current CPU's. A sketch of the GPU architecture pipeline is illustrated in figure 1. And third, the hardware is laid out in an extremely parallelized fashion exploiting the above pipeline limitations. For example, current consumer low-cost graphics cards are designed with eight processor units working completely independent in parallel.

The instruction set is very restricted in comparison to current CPU's, but highly optimized. Most instructions are executed within a single clock cycle.

In the past, the graphics processor instruction set was too limited to be freely programmable. It was known as fixed function pipeline processors with very limited programmability.

Current (fourth generation) graphics architecture fulfills full programmability by true branching and looping capabilities. A detailed technical description of current GPU hardware is beyond the scope of this paper. We refer to the technical specifications of the main manufacturers [1, 10].

But it is important to understand the concept of four main stages illustrated in figure 1.

1. Vertex processing. Input is one vertex with a restricted set of associated per vertex parameters. For example per vertex color or texture coordinate and additional parameters constant for all vertices like scalars, vectors and texture maps. Every input vertex generates exactly one output vertex controlled by a vertex program.

2. The Rasterizer is interpolating tessellated vertices and associated per vertex data linearly into fragments. These fragment are potential pixels for a given viewing frustum projection. This is a very efficient (bi-)linear interpolator used for barycentric weighting, for example.

3. The Fragment processors input are the fragments generated by the rasterizer. A fragment can be thought of as a potential pixel on screen manipulated by a fragment program. Programmability and parameters for vertex-and fragment program are comparable.

4. Final stage selects one fragment projected into each screen pixel (or rejects all fragments). For example the nearest fragment could be selected by its depth value.

Please note that one input vertex is generating exactly one output vertex and one input-fragment is generating exactly one output-fragment. All vertices (and fragments) are processed in parallel and independently of any other. No order of processing is guaranteed. In particular each screen pixel is generated independently of its spatial neighbors.

Current hardware supports freely programmable vertex and fragment processing but the rasterizer is a fixed function unit supporting only linear interpolation between support points defined by vertices and primitives such as triangles, lines and points of a specified geometry.

For highest performance one should avoid too heavy computations in the fragment processor. All operations that can be linearly interpolated by the rasterizer should be moved to the vertex processor for most efficient use of the pipeline.

In our case of a high number of fragments (potential pixels) and a smaller number of vertices (triangles) the fragment processor is usually limiting the pipeline performance.

Current graphics cards have a very wide memory bus (AGP) to transfer data from main memory to graphics memory, but only a small bus (PCI) to transfer back. However, we expect this bottleneck to be solved with the availability of the new PCI Express Bus capable visual processors.

4 Our Method

We address the problem of assigning each pixel of a reference camera a depth (or 3D scene point), known as the stereo problem. Stereo algorithms can be divided into four major steps [13]

1. computing a per pixel matching cost (e.g. for pixel pairs),
2. aggregation of cost over support region (e.g. 3×3),
3. disparity or depth computation based on selection and optimization,
4. refinement and post-processing.

In this contribution we address the multi-stereo problem for multiple color input images in a known but arbitrary camera configuration. The above stages describe the two-view case. For the multi-view case we introduce an additional multi-camera scoring stage which exploits multi-view constraints described in section 4.4.

Please note that we do neither need rectified images nor a fixed internal calibration or equal image sizes. We follow the idea of a generalized sweep approach to compute a depth map for a reference camera. Any virtual camera may be used as reference, in particular one of the input real cameras.

4.1 Projection

Each camera can be described by its mapping from 3D world coordinates to 2D image coordinates. The mapping of a scene point X into an image point x by a camera can be modeled by a projection matrix P as

$$x = PX. \tag{1}$$

The corresponding intensity values $I_{(X,i)}$ for all cameras P_i with associated image textures tex_i can be obtained by standard projective texture lookup as

$$I_{(X,i)} = tex\left(P_i X\right) \tag{2}$$

Our task is finding the most probable depth $d = |X - C|_{L2}$ for a pixel x on a ray through the pixel x itself and a reference cameras projection center C. Projective texture mapping of the vertex processor and projective texture lookup in the fragment program is performed very efficiently in hardware.

4.2 Matching Cost

If there is a scene surface at X, then the intensities $I_{(X,i)}$ are similar for all cameras P_i, known as the photo-consistency constraint. This holds at least for diffuse, non-occluded scene points, as we assume. We use the squared intensity difference (SD) to measure the dissimilarity between two intensity scalars s_k,s_l

$$SD\left(s_k, s_l\right) = \left(s_k - s_l\right)^2. \tag{3}$$

The scalar dissimilarity measure for a pair of color values $a = (a_0, a_1, a_2)$, $b = (b_0, b_1, b_2)$ can be computed as sum of SD over all channels

$$SSD\left(a, b\right) = \sum_{c \in channel} SD\left(a_c, b_c\right). \tag{4}$$

This local measure with a small 1×1 support of only one color value is susceptible to noise and lacks discrimination in texture-less regions.

Yang et. al. compensated this partially by comparing a base camera against multiple destination cameras by summation of the SSD pairs (SSSD) [15]. We extend this approach without support region increase by a customized multi-camera scoring described in section 4.4.

In addition we use truncated SSD scores (TSSD) to limit the influence of outliers due to image noise and non-diffuse surfaces. The SNR can be adapted locally but for state-of the art cameras it is sufficient to assume a small a priori known SNR. Overflow problems are avoided by this thresholding, too.

$$TSSD\left(a, b\right) = min\left(thresh_{SNR}, SSD\left(a, b\right)\right) \tag{5}$$

In addition spatial aggregation of the dissimilarity can be used to make the estimation more robust at the cost of additional rendering passes.

4.3 Spatial aggregation over support region

Summation of the matching cost in a window or a weighted summation for example by a Gaussian filter 'smearing' can be used to aggregate spatial neighboring dissimilarity measures into one more robust score. This is equivalent to filtering with a box- or Gaussian 2D filter in the dissimilarity image. Yang followed this approach for the two-view case [14] with an approximation of a symmetrical

Gaussian filter centered on the reference position x. An approximation of this filtering can be computed efficiently in hardware by summation over multiple levels of the dissimilarity images mipmap pyramid (MML). However, it is important to point out that the mipmap box filter is applied correctly only on modulo power-of-two coordinates (2^{level}) for each mipmap level. Their approximative Gaussian depends implicitly on the pixel position. The support region is slightly shifted around each pixel depending on the pixels distance to next power of two position.

It is also important to note that aggregation by a centroid filter implicitly assumes piecewise continuous frontoparallel scene surfaces. Slanted surfaces and depth discontinuities are not handled correctly. In addition the creation and summation of mipmap levels in hardware is computationally not for free. It is efficient only because of decreasing number of pixels with every pyramid with a maximum total overhead of 33 % extra pixel for a complete pyramid.

We follow a different approach to make depth estimation more robust. Instead of centroidal spatial 'smearing' SSD scores by support region filtering we follow a best n of m multi camera selection scheme and the idea of of spatially shiftable windows with approximative correct depth discontinuity handling similar to the CPU implementations of Kang [7] and Hirschmueller [4].

The idea is that the scene is continuous in at least one of the multiple windows around the central point, see figure 2. This can be computed very efficiently in hardware by summation of four values with a single bilinear texture lookup shifted from x with an offset of $\sqrt{\frac{1}{2}}$ pixel in diagonal directions. This idea can be used for arbitrary size aggregation windows in additional rendering passes. An arbitrary size $2n \times 2m$ window can be summed with nm bilinear texture lookups plus arithmetic instructions for the summation.

Please note that our approach is neither limited to power-of-two texture size nor coordinates, because we do not need recursive box-filtering by mipmapping. We can handle rectangle textures of arbitrary size[1] and avoid extra overhead for padding arbitrary size to next power-of-two size.

[1]limited by hardware and video memory (currently max. 4096×4096.)

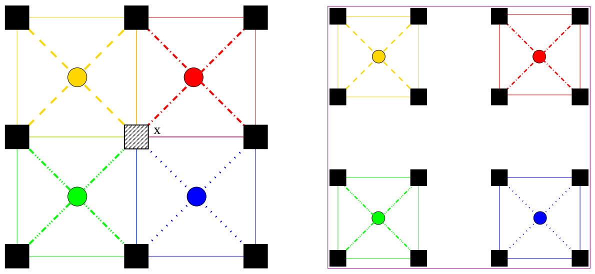

Figure 2: SSD is computed at positions marked by squares. Left: Four shiftable windows in 8-neighborhood of center x can be summed very efficiently independently by bilinear texture lookup at diagonal positions marked by circles. Depth discontinuities horizontal,vertical and diagonal are handled correctly by at least one of the windows. Right: Arbitrary $2n \times 2m$ windows are summed by nm bilinear texture lookups .

The assumption of continuous surface in one of the diagonal directions is an approximative simplification for occlusion handling. But it can be computed without major performance loss and is handling occlusions better than the original approach.

For the two-view case right-left consistency testing may be used to reduce mismatches. However, currently we do not follow this approach because it is an expensive operation which in particular does not scale well for for the multi-view case. We can handle occluded regions only approximately correctly, but we overcome semi-occluded regions visible in at least two cameras using more than two cameras. The main goal of this approach is keeping depth discontinuities sharp. Halo-effects and depth blurring due to the size of the support region is avoided.

Different viewpoints of the cameras result in affine distortions of the support region of the different cameras. Small baseline and small support regions can be used to limit this effect. We use a different approach to improve the estimation without support region increase which we describe in the next sections.

One should notice that current GPU architecture can handle only two texture accesses per texture unit in a single cycle. Support regions of arbitrary size can be handled, but at the cost of performance. For maximum performance we can substitute spatial by temporal aggregation over the cameras described in section 4.4. All depth hypotheses for all rays can be handled independently in a single rendering pass to avoid memory transfer even within the GPU memory. In particular we do not need intermediate rendering to a frame buffer or a texture for all depth hypothesis. The complete result is rendered in a single but more complex rendering pass for maximum performance.

However, this 1-pass approach is perfectly suitable only for 1×1 support regions because dissimilarity computations could not be shared between fragments and may be computed twice for adjacent fragments.

4.4 Multi-camera scoring stage

Multiple images can be used to enhance estimation without increasing support region size and simultaneously accounting for occlusions and noise. Simple summation of the (aggregated) TSSD scores is only valid for regions of the scene being visible in all cameras.

Instead of scoring only a reference camera against all other cameras as done in [15] we are scoring all $\binom{2}{n}$ pixel pairs within n images. For example four images are scored by all 6 pixel-pairs instead of only three. Please note that the number of texture lookups remains the same. Only slightly more arithmetic instructions are required by our approach.

Instead of just summing these scores we follow a statistical selection scheme to pick only the best scores and sort out outliers immediately. We follow two real-time-capable approaches:

- best n of m scores sorting out worst $m - n$ scores considered as mismatches due to occlusions or specular surfaces,
- best selection of left and right half sequence for a central camera. If the cameras are approx. on a line left and right scores are handled separately. Half sequence scores are compared and the best one is chosen assuming occlusions occur only in one of the half-sequences.

4.5 Handling partial overlap

Partial overlap of different field of view has to be handled explicitly for general camera configurations because a ray of depth hypotheses may intersect different field of views (see figure 3). We are optionally normalizing the summed multi-camera scores with the number of pairs where a scene point is visible for each depth hypothesis. This is important because a ray of depth hypotheses may intersect different field of views resulting in inadequate matching scores which have to be normalized to be comparable among the whole ray. Test for visibility can be executed very efficiently in hardware by exploiting clamping with a special marker value. Instead of checking whether texture coordinates (x, y) are within valid range (e.g. $[0..1]$) we use a special border color $(0, 0, 0)$ to identify whether a texture lookup was clamped. For each hypothesis (x, y, d) we pre-compute the number of cameras vis where the associated relative 3D point is within field of view and store it as lookup texture. Summation of visible pairs is normalized by vis very efficiently.

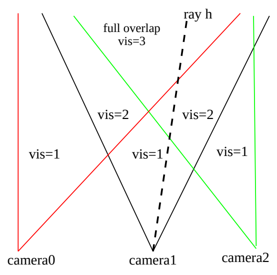

Figure 3: Field of view of three cameras overlaps only partially. Normalization depending on vis is used to normalizes matching scores for comparison on a ray of depth hypotheses h.

We follow an additional second approach of simplifying this approach further for maximum performance exploiting our dissimilarity measure TSSD. Images are initialized such that texture access outside an image returns a different but constant value for each camera. The set of n border values for n cameras is chosen so that all SD pairs between border values are bigger than the truncation threshold. This is possible for small truncation thresholds and a limited number of cameras. Sum of all non-clamped values is normalized by the visibility of the hypothesis. Partial overlap is handled as perfect mismatch which is wrong because we have no information of the dissimilarity. This is not correct but very efficient and the errors introduced are limited by our best n of m selection scheme and the truncation threshold.

4.6 Selection and Post-processing

The best depth hypothesis on a ray is selected in hardware almost for free by manipulating a fragments depth value. Matching score of each fragment (x, y, d) is set as depth value and best hypothesis selected in hardware by depth-compare. All matches beyond a required maximum dissimilarity threshold are invalidated for free by setting an appropriate z-range.

Following this approach the resulting depth maps are good enough for pre-visualization. Remaining salt and pepper noise can be removed by local median filtering at the cost of performance if desired.

4.7 Post processing and Consistency Testing

The resulting depth map is significantly perturbed by error noise for small support regions. We use statistical and heuristic post-processing to increase depth map quality preserving depth discontinuities. The errors can be categorized into

- statistical errors due to image noise,
- matching ambiguities, in particular for texture-less regions and repeated textures,
- systematical errors due to our approximations and wrong hypotheses.

Noise can be removed for more satisfactory visualization easily by spatially filtering the depth map at the cost of small details. In particular salt and pepper noise of one-pixel obstacles can be removed completely by local median filtering.

Handling matching ambiguities in a general fashion is hardly possible for our local approach. We try to solve it by using different baselines and orientation between the cameras to make ambiguities due to repeated textures occurring simultaneously in many camera pairs very unlikely.

In addition consistency testing can be used to invalidate uncertain depth values. For every depth value associated to a ray of depth hypotheses we count the number of cameras in which the voxel is occluded. We follow a shadow-mapping approach to threshold the visibility of the depth hypothesis to at least two cameras. Both, the occluded and the occluding pixel can be invalidated as inconsistent. We follow a shadow-mapping approach to invalidate uncertain estimations very efficiently in hardware.

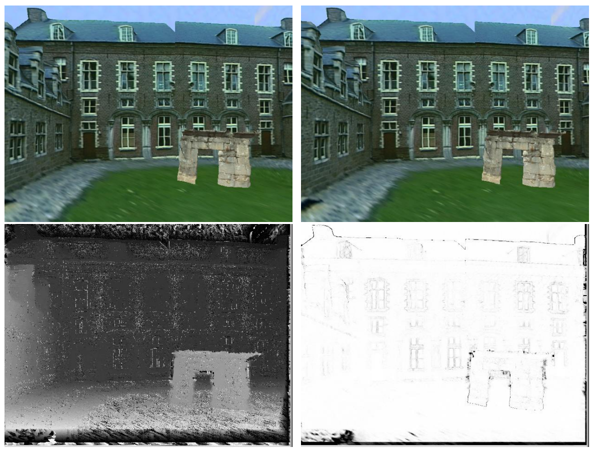
Figure 4: Top: Two of four input images of the 'castle' sequence. Bottom left: Depth map estimated in real-time with 'best 5 of 6 TSSD' algorithm and winner-take-all selection. Bottom right figure illustrates the remaining dissimilarity. Dark values=high dissimilarity, light values=low dissimilarity.

5 Results

Our prototype system for real-time capture and pre-visualization was implemented for four digital fire-wire cameras mounted on a pole. We used a mid-range Nvidia Geforce FX 5600 Graphics processor which has only four parallel texture units. It is embedded in a standard PC with a Pentium 4 main processor, an AGP 8x bus and 1GB of RAM. Please note that the performance scales linearly with the parallelism of higher performance graphics cards with eight or more texture units which are available, now. Please note that rectified images are not necessary. Radial lens distortion can be compensated by dependant texture lookup but wasn't used in the following experiments.

Performance evaluation on a synthetic test scene is evaluated in next section. Results for a natural scene of highly complex geometry is demonstrated in section 5.2.

5.1 Synthetic test sequence

We used a synthetic test sequence of four images for performance evaluation. The scene consists of a foreground portal occluding a background building and low-textured grass floor. Dense depth is estimated from four images in arbitrary configuration. See figure 4 for two of four input images and qualitative raw estimation result computed with 'best 5 of 6' camera selection, TSSD matching cost on a 1×1 support and shiftable windows. Neither spatial aggregation nor post-processing is used.

image size	read-back time	rendering time
320×240	2 ms	60 ms
640×480	7 ms	210 ms
704×576	9 ms	280 ms
1280×960	28 ms	760 ms

Table 1: Rendering time and overhead scale linearly with image size for (20 depth hypotheses). Read-back includes overhead to transfer images from main memory to graphics card and result back to main memory.

scoring algorithm	render time
classic SSSD for 3 pairs	39 ms
best 2 of 3 TSSD	44 ms
SSD for 6 pairs	78 ms
best 5 of 6 TSSD	94 ms

Table 2: Performance comparison of SSSD and best n of m TSSD scoring algorithms comparing for 3 and 6 SD pairs for four 320×240 img.

A quantitative performance evaluation of rendering time depending on search space and image dimensions for four cameras is given in table 1. Rendering time (or frame rate) scales linearly with image resolution and search space, as expected (figure 5).

5.2 Dinosaur scene

We evaluated our system with a natural scene suitable for image based rendering [8]. The scene consists of a foreground dinosaur skeleton and background arches. It is captured with a four-camera rig (see figure 6). Please note the high complexity of the scene with lots of occlusions, non-diffuse and texture-less surfaces. We used a search space of 20 depth hypotheses evaluated by our 'best half-sequence' approach on a 1×1 support with TSSD scoring and winner-take-all selection. Processing time of the four 704×576 images is 280 ms.

Figure 7 (bottom left) illustrates the real-time computed depth estimation. Most of the present salt and pepper noise can be removed by post-processing, if desired.

The selected best half-sequence of our best n of m approach is illustrated in figure 7, bottom right. Pixels with significantly better score (min. 10 % better) in one of its half sequences are marked.

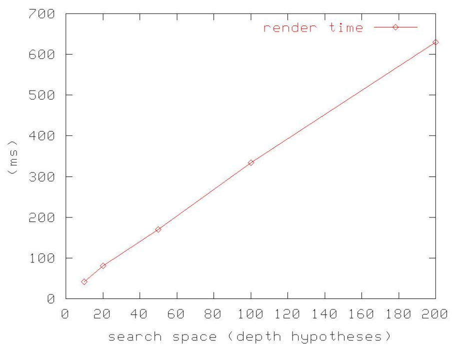

Figure 5: Multi-stereo rendering time (ms) vs. depth search space dimension for four 320×240 input images using best 2 of 3 TSSD approach.

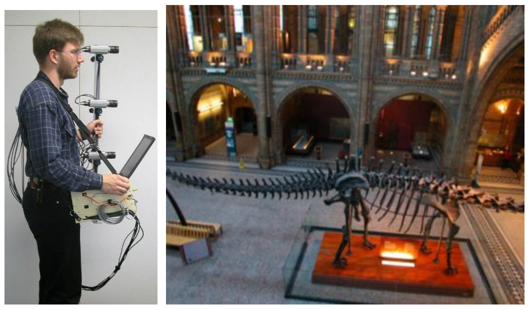

Figure 6: Left: portable multi camera image capture system, Right: Overview of the dinosaur scene being reconstructed.

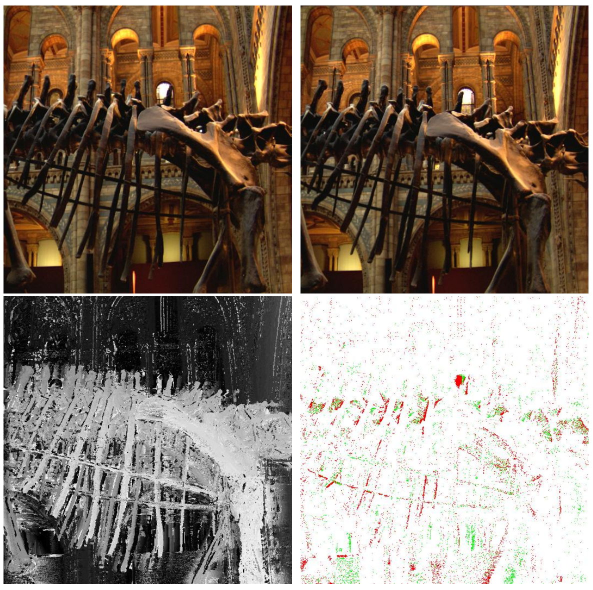
Figure 7: Top: two of four input images. Bottom Left: estimated depth. Brighter values indicate nearer scene points. Bottom Right: Best half sequence selection. red indicates best left, green best right score. white means no significant difference.

Red and green pixel mean left or right half sequence is significantly better than any other half sequence. This is a measure for high depth discontinuity probability. Pixels with half sequence discrepancy smaller 10 % are marked white.

6 Conclusions

We presented an algorithm for dense depth estimation from multiple images. Our implementation is running almost entirely on standard graphics hardware. Depth is estimated in real-time with up to 20 fps from four images.

Live depth estimation can be used to adapt the camera path during image acquisition to ensure all desired scene details are captured with sufficient precision. 3D scene capture is simplified and unsuccessful image acquisition avoided. Capture quality and cost is improved.

Our implementation uses the higher level shader language Cg [10]. First experiments showed further performance potential by hand optimized shader code.

A quantitative comparison of hand-optimized GPU and CPU implementations is desirable. More performance could be reached by parallel usage of CPU and GPU.

Our current systems uses a calibrated four camera rig. Further work will focus on integrating real-time camera path estimation to be able to change relative camera calibration during capture. Then our system could be easily extended to build a complete global scene model on the fly.

Acknowledgments

This work is being funded by the European project IST-2000-28436 ORIGAMI of the European Community. Images of the dinosaur scene were supplied by Oliver Grau, BBC Research.

References

[1] Ati technologies inc. www.ati.com.

[2] Robert T. Collins. A space-sweep approach to true multi-image matching. In *Proc. Computer Vision and Pattern Recognition Conf.*, pages 358–363, 1996.

[3] L. Falkenhagen. Depth estimation from stereoscopic image pairs assuming piecewise continuous surfaces. In *Proc. of European Workshop on combined Real and Synthetic Image Processing for Broadcast and Video Production, Hamburg*, 1994.

[4] Heiko Hirschmueller. Improvements in real-time correlation-based stereo vision. In *Proc. of IEEE Workshop on Stereo and Multi-Baseline Vision, Kauai, Hawaii*, 2001.

[5] S. L. Hingorani I. J. Cox and S. B. Rao. A maximum likelihood stereo algorithm. In *Computer Vision and Image Understanding*, volume 63 (3), pages 542–567, 1996.

[6] Intel corp. www.intel.com, 2003.

[7] S.B. Kang, R. Szeliski, and J. Chai. Handling occlusions in dense multiview stereo. In *Proc. of IEEE Conference on Computer Vision and Pattern Recognition*, Dec. 2001.

[8] R. Koch, J.M. Frahm, J.F. Evers-Senne, and J. Woetzel. Plenoptic modeling of 3d scenes with a sensor-augmented multi-camera rig. In *Proc. of Tyrrhenian International Workshop on Digital Communication (IWDC)*, Sept. 2002.

[9] Vladimir Kolmogorov and Ramin Zabih. Multi-camera scene reconstruction via graph cuts. In *Proc. of ECCV*, pages 82–96, 2002.

[10] Nvidia corp. www.nvidia.com.

[11] Nabil Boukala P. Colantoni and Jerome Da Rugna. Fast and accurate color image processing using 3d graphics cards. In *Proc. of 8th International Fall Workshop: Vision Modeling and Visualization, Munich, Germany*, Nov. 2003.

[12] Point grey research inc. www.ptgrey.com.

[13] D. Scharstein, R. Szeliski, and R. Zabih. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. In *Proc. of IEEE Workshop on Stereo and Multi-Baseline Vision, Kauai, HI*, 2001.

[14] Ruigang Yang and Marc Pollefeys. Multi-resolution real-time stereo on commodity graphics hardware. In *Proc. of Conference on Computer Vision and Pattern Recognition (CVPR '03), Madison, Wisconsin*, June 2003.

[15] Ruigang Yang, Greg Welch, and Gary Bishop. Real-time consensus-based scene reconstruction using commodity graphics hardware. In *Proc. of Pacific Graphics, Tsinghua University, Beijing, China*, October 2002.

[16] Christopher Zach, Andreas Klaus, Markus Hadwiger, and Konrad Karner. Accurate dense stereo reconstruction using graphics hardware. Technical Report TR 2003 018, VRVis, Vienna, Austria, June 2003.

[17] Christopher Zach, Andreas Klaus, Bernhard Reitinger, and Konrad Karner. Optimized stereo reconstruction using 3d graphics hardware. Technical Report TR 2003 024, VRVis, Vienna, Austria, August 2003.