A parallel implementation of 3D Zernike moment analysis

Daniel Berjón, Sergio Arnaldo, Francisco Morán

Grupo de Tratamiento de Imágenes, Universidad Politécnica de Madrid, Spain

ABSTRACT

Zernike polynomials are a well known set of functions that find many applications in image or pattern characterization because they allow to construct shape descriptors that are invariant against translations, rotations or scale changes. The concepts behind them can be extended to higher dimension spaces, making them also fit to describe volumetric data. They have been less used than their properties might suggest due to their high computational cost.

We present a parallel implementation of 3D Zernike moments analysis, written in C with CUDA extensions, which makes it practical to employ Zernike descriptors in interactive applications, yielding a performance of several frames per second in voxel datasets about 200^3 in size.

In our contribution, we describe the challenges of implementing 3D Zernike analysis in a general-purpose GPU. These include how to deal with numerical inaccuracies, due to the high precision demands of the algorithm, or how to deal with the high volume of input data so that it does not become a bottleneck for the system.

Keywords: Zernike moments, geometric moments, parallel programming, CUDA, GPU, implementation

1. INTRODUCTION

Classical Zernike two-dimensional polynomials are an orthogonal set of functions in the unit disk. They are formulated in polar terms as a product of separate angular and radial components. The angular factors are themselves an orthogonal set of functions in $[0, 2\pi)$ and the radial factors are whatever family of functions it takes to make the product an orthogonal set in the unit disk. The usual angular functions of choice to analyse real functions are sines and cosines, which require the radial modulation factors to be made of complicated gaussian hypergeometric functions. This polar formulation of the Zernike polynomials is both interesting, because it lends itself to easily build moments which are invariant against rotations around the origin, and inconvenient, because data for performing numeric computations are usually sampled using square lattices.

The high cost of computing the hypergeometric functions and resampling data has favoured the use of simpler alternatives such as Hu's moment invariants[1] to provide rotation-invariant shape descriptors. However, Zernike polynomials do provide valuable insight about the structure of the shape to be analysed. Conceptually, they serve the same purpose as 2D Fourier series, each of the functions in the set encoding different spatial frequencies in the radial or angular directions. Unsurprisingly, these features caught the attention of the image processing community long ago[2,3] and many attempts have been made at reducing their computational cost in order to make them practical.[4–6] However, it has been only very recently that the availability of commodity parallel hardware and GPGPU (general-purpose GPU) programming tools have made possible to compute them at interactive frame rates.[7]

The basic idea and structure of Zernike polynomials is not only valid in two dimensions, but theoretically also in higher-dimensional spaces. In particular, Canterakis formulated Zernike polynomials for the three-dimensional case,[8] in which the angular factor are spherical harmonics. Spherical harmonics are a well known basis on the sphere and the moments of a function projected on them can be arranged to form rotation-invariant descriptors of functions defined on the unit sphere.[9] Since the only dependency of 3D Zernike polynomials on angles is indeed the spherical harmonics function set, this property also applies. However, computation of 3D Zernike moments is even more expensive than that of classical 2D Zernike moments. The same issues that arise in those (resampling costs, complexity of the radial function) remain, only worse because there are more samples, and now also the angular functions are complex in more than one sense.

Though there is obvious interest and research activity in shape descriptors for 3D objects,[10] not many attempts have been made at accelerating the computation of these 3D Zernike moments. Notably, Novotni and Klein were able to

Contact e-mail: {dbd,sad,fmb}@gti.ssr.upm.es

Parallel Processing for Imaging Applications, edited by John D. Owens, I-Jong Lin, Yu-Jin Zhang, Giordano B. Beretta,
Proc. of SPIE-IS&T Electronic Imaging, SPIE Vol. 7872, 787209 · © 2011 SPIE-IS&T · CCC code: 0277-786X/10/$18
doi: 10.1117/12.876683

build upon Canterakis' work and reformulate the 3D Zernike polynomials as linear combinations of geometric moments, thus bypassing the resampling phase and moving much complexity offline.[11] New algorithms to compute both exact and approximate moments from triangle meshes have been presented very recently.[12] The implementation we present in this paper is an adaptation of Novotni and Klein's algorithm for GPUs which makes it suitable for interactive applications.

2. 3D ZERNIKE MOMENTS

In this section we will attempt to summarise the results of Novotni and Klein and their formulation of 3D Zernike moments, since we will use these concepts to build our parallel implementation. Please refer to the original paper for a complete treatment of the theoretical background.

The orthogonal set of 3D Zernike functions Z_{nl}^m is defined as:

$$Z_{nl}^m(\rho,\theta,\phi) = R_{nl}(\rho)\cdot Y_l^m(\theta,\phi) \tag{1}$$

for $\rho \in [0,1], \theta \in [0,\pi], \phi \in [0,2\pi)$. Indices designing each of the member functions of the set follow these rules: n designates the main order of the function and must be a non-negative integer; l must be a non-negative integer restricted to $l \leq n$ and $n-l$ must be an even number; m must be an integer ranging from $-l$ to l.

3D Zernike moments are defined as:

$$\Omega_{nl}^m = \frac{3}{4\pi}\iiint_{\|\boldsymbol{x}\|\leq 1} f(\boldsymbol{x})\,\overline{Z_{nl}^m(\boldsymbol{x})}d\boldsymbol{x} \tag{2}$$

The radial portion of the 3D Zernike polynomials is provided by the spherical harmonics set of functions, which are defined as:

$$Y_l^m(\theta,\phi) = N_l^m P_l^m(\cos\theta)\,e^{im\phi} \tag{3}$$

where N_l^m is the normalisation factor

$$N_l^m = \sqrt{\frac{2l+1}{4\pi}\frac{(l-m)!}{(l+m)!}} \tag{4}$$

and P_l^m is the associated Legendre function.

Canterakis showed that 3D Zernike functions can be rewritten as homogeneous polynomials in cartesian coordinates using harmonic polynomials for the derivation. Harmonic polynomials e_l^m are defined as:

$$e_l^m(\boldsymbol{x}) = r^l Y_l^m(\theta,\phi) \tag{5}$$

Expanding the associated Legendre functions and converting into cartesian coordinates, this can be rewritten as

$$e_l^m(\boldsymbol{x}) = c_l^m|\boldsymbol{x}|^l\left(\frac{ix-y}{2}\right)^m z^{l-m}\sum_{\mu=0}^{\lfloor\frac{l-m}{2}\rfloor}\binom{l}{\mu}\binom{l-\mu}{m+\mu}\left(-\frac{x^2+y^2}{4z^2}\right)^\mu \tag{6}$$

where c_l^m is the normalisation factor

$$c_l^m = c_l^{-m} = \frac{\sqrt{(2l+1)(l+m)!(l-m)!}}{l!}$$

The above definitions apply to indices $m \geq 0$. For $m < 0$ the following relation applies:

$$e_l^{-m}(\boldsymbol{x}) = (-1)^m\,\overline{e_l^m(\boldsymbol{x})} \tag{7}$$

As for the radial part of the equation, by definition has to be a function that, when multiplied by the spherical harmonics that we selected as angular factor, satisfies the orthonormality criterion

$$\frac{3}{4\pi}\iiint_{\|\boldsymbol{x}\|\leq 1} Z_{nl}^m(\boldsymbol{x})\,\overline{Z_{n'l'}^{m'}(\boldsymbol{x})}d\boldsymbol{x} = \delta_{nn'}\delta_{ll'}\delta_{mm'} \tag{8}$$

and can be found to be

$$R_{nl}(r) = r^l \sum_{\nu=0}^{k} q_{kl}^{\nu} r^{2\nu} \tag{9}$$

where $k = \frac{n-l}{2}$ and the coefficients q_{kl}^{ν} are:

$$q_{kl}^{\nu} = \frac{(-1)^k}{2^{2k}} \sqrt{\frac{2l+4k+3}{3}} \binom{2k}{k} (-1)^{\nu} \frac{\binom{k}{\nu}\binom{2(k+l+\nu)+1}{2k}}{\binom{k+l+\nu}{k}} \tag{10}$$

Therefore, combining equations 1, 5 and 9 the 3D Zernike polynomials can be written in cartesian coordinates:

$$Z_{nl}^{m}(\boldsymbol{x}) = \sum_{\nu=0}^{k} q_{kl}^{\nu} |\boldsymbol{x}|^{2\nu} e_l^m(\boldsymbol{x}) \tag{11}$$

which implies, considering that m indices affect only the angular portion of the function and applying equation 7,

$$Z_{nl}^{-m}(\boldsymbol{x}) = (-1)^m \overline{Z_{nl}^{m}(\boldsymbol{x})} \tag{12}$$

and, consequently, for the 3D Zernike moments of a particular function,

$$\Omega_{nl}^{-m} = (-1)^m \overline{\Omega_{nl}^{m}} \tag{13}$$

Combining equation 11 with equation 6 and expanding results in:

$$\begin{aligned} Z_{nl}^{m}(\boldsymbol{x}) \quad = \quad & c_l^m 2^{-m} \sum_{\nu=0}^{k} q_{kl}^{\nu} \sum_{\alpha=0}^{\nu} \binom{\nu}{\alpha} \\ & \cdot \sum_{\beta=0}^{\nu-\alpha} \binom{\nu-\alpha}{\beta} \sum_{u=0}^{m} (-1)^{m-u} \binom{m}{u} i^u \\ & \cdot \sum_{\mu=0}^{\lfloor \frac{l-m}{2} \rfloor} (-1)^{\mu} 2^{-2\mu} \binom{l}{\mu} \binom{l-\mu}{m+\mu} \sum_{v=0}^{\mu} \binom{\mu}{v} \\ & \cdot x^{2(v+\alpha)+u} y^{2(\mu-v+\beta)+m-u} z^{2(\nu-\alpha-\beta-\mu)+l-m} \end{aligned} \tag{14}$$

If the substitutions $r = 2(v+\alpha)+u$, $s = 2(\mu-v+\beta)+m-u$ and $t = 2(\nu-\alpha-\beta-\mu)+l-m$ are defined (note that $r+s+t = 2\nu+l \leq 2k+l = n$), it is possible to define

$$\begin{aligned} \sum_{r+s+t \leq n} \chi_{nlm}^{rst} \quad = \quad & c_l^m 2^{-m} \sum_{\nu=0}^{k} q_{kl}^{\nu} \sum_{\alpha=0}^{\nu} \binom{\nu}{\alpha} \\ & \cdot \sum_{\beta=0}^{\nu-\alpha} \binom{\nu-\alpha}{\beta} \sum_{u=0}^{m} (-1)^{m-u} \binom{m}{u} i^u \\ & \cdot \sum_{\mu=0}^{\lfloor \frac{l-m}{2} \rfloor} (-1)^{\mu} 2^{-2\mu} \binom{l}{\mu} \binom{l-\mu}{m+\mu} \sum_{v=0}^{\mu} \binom{\mu}{v} \end{aligned} \tag{15}$$

where each χ_{nlm}^{rst} is defined as those terms from the right hand sums that satisfy its particular combination of indices (for a given (n,l,m) tuple, many χ_{nlm}^{rst} will be zero). Then the 3D Zernike polynomials can be written as:

$$Z_{nl}^{m}(\boldsymbol{x}) = \sum_{r+s+t \leq n} \chi_{nlm}^{rst} \cdot x^r y^s z^t \tag{16}$$

and the 3D Zernike moments can be rewritten as:

$$\Omega_{nl}^{m} = \frac{3}{4\pi} \sum_{r+s+t \leq n} \chi_{nlm}^{rst} \iiint_{\|\boldsymbol{x}\| \leq 1} f(\boldsymbol{x})\, x^r y^s z^t d\boldsymbol{x} \tag{17}$$

where the integral part of the equation is the definition of 3D geometric moments *in continuous space*. Note that thanks to this formulation, only the geometric moments need to be computed for each new function; all the χ_{nlm}^{rst} coefficients may be very expensive to obtain, but need only be found once, off-line.

3. PARALLEL COMPUTATION OF 3D ZERNIKE MOMENTS

In this section, we will explain how we adapted and implemented the algorithm described in the preceding section to a GPU using the CUDA extensions for C. Our implementation is designed to work with voxel binary data, this is, each voxel is either free or occupied, and we assume that the voxel grid is sized $N \times N \times N$, which is frequently the case.

3.1 Data normalisation

We want to describe data in a way which is invariant to translations and scale changes, so we must first normalise the input dataset. In order to normalise it against translations, we first need to find a stable reference point in the shape described by the occupied voxels in the grid to place the origin of coordinates. As we understand the problem, this reference point should be located in the same position with respect to the shape, regardless of whether it is rotated or scaled, and despite the noise introduced by the boundary discretization.

The easiest and computationally cheapest choice for a central tendency is the centroid, which can be defined by its geometric properties as:

$$\underset{\boldsymbol{y} \in \mathbb{R}^3}{\arg\min} \sum_i \|\boldsymbol{x}_i - \boldsymbol{y}\|^2 \tag{18}$$

Discrepancies between different scaled and/or rotated versions of the same shape are expected to be located on the boundaries of the object, this is, away from the centre. Since the centroid position is more influenced by far samples than it is by near samples, we thought that it could be not stable enough and evaluated the geometric median, which is defined as:

$$\underset{\boldsymbol{y} \in \mathbb{R}^3}{\arg\min} \sum_i \|\boldsymbol{x}_i - \boldsymbol{y}\| \tag{19}$$

However, whereas the centroid is very easy to calculate, just by averaging the projections of the samples onto the directions of each axis independently, there is no known closed form to calculate the geometric median, and costly iterative procedures must be used.[13]

In the end, our tests indicated that, at least for the kind of shapes we cared about and tried both methods on (mostly human figures in different poses), there is no significant difference between them, yielding points within very few voxels from each other at the most. Therefore, we chose the centroid as reference point for the rest of the calculations, with an initial implementation on CPU, since the bulk of the cost of the Zernike moments is the computation of the geometric moments and the expected speed-up is not very big because the computation must yield a single result, which means concurrent data access.

Once the location of the new origin of coordinates is determined, some scale factor must be found to relate the integer coordinates of the voxel grid to coordinates in $\mathbb{R}^3$, so that the shape under study fits into the unit ball, which is the domain under the Zernike polynomials are well defined. For reasons that will be apparent when studying the computation of the geometric moments, we will interpret each voxel as a solid cube with edge size one, its center sitting on the integer coordinates that correspond to its indices in the array that represents the voxel grid.

Statistical measures of the radius of the object are not good enough because, depending on the general shape of the object, some portions of it could lie beyond $\|\boldsymbol{x}\| \leq 1$, so we just assume the dataset to be free of outliers and measure its real radius from the centroid $\boldsymbol{c}$:

$$R = \max \left\| \begin{pmatrix} \|i - c_x\| + \frac{1}{2} \\ \|j - c_y\| + \frac{1}{2} \\ \|k - c_z\| + \frac{1}{2} \end{pmatrix} \right\| \quad (i,j,k) \in V \tag{20}$$

where V is the set of all occupied voxels. Thus, we can establish a correspondence between integer indices and real coordinates of the centres of the voxels:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \operatorname{norm}(i,j,k) = \frac{1}{R}\begin{pmatrix} i-c_x \\ j-c_y \\ k-c_z \end{pmatrix} \tag{21}$$

Again, the measurement of the radius of the object is carried out on the CPU because its computation time is negligible compared to that of the geometric moments and the expected speed-up is small: the comparisons between the maximum value at some point and any candidate to replace it must be atomic if implemented concurrently to avoid race conditions.

3.2 Geometric moments

On equation 17, Zernike moments are redefined as linear combination of geometric moments. If we combine that definition with the interpretation of voxel data we have made in the previous subsection, $f(\boldsymbol{x})$ is a piecewise-constant function and we can write:

$$\iiint_{\|\boldsymbol{x}\|\leq 1} f(\boldsymbol{x})\, x^r y^s z^t d\boldsymbol{x} = \sum_{v\in V} \iiint_v x^r y^s z^t d\boldsymbol{x} \tag{22}$$

We can easily solve the integral for each voxel:

$$\iiint_v x^r y^s z^t d\boldsymbol{x} = \int_{\frac{i-c_x-\frac{1}{2}}{R}}^{\frac{i-c_x+\frac{1}{2}}{R}} x^r dx \cdot \int_{\frac{j-c_y-\frac{1}{2}}{R}}^{\frac{j-c_y+\frac{1}{2}}{R}} y^s dy \cdot \int_{\frac{k-c_z-\frac{1}{2}}{R}}^{\frac{k-c_z+\frac{1}{2}}{R}} z^t dz \tag{23}$$

but we need not perform one integral per voxel: all voxels with the same i, j or k share the same sub-integrals on x, y and z respectively, so we can build a lookup table of dimensions $3 \times n_{max} \times N$, where n_{max} is the maximum n up to which we want to compute the Zernike moments. Ultimately this is limited by the availability of precomputed χ_{nlm}^{rst} coefficients (in our case, we computed them up to order 20 using Maple). Again, we compute the lookup table in CPU because this effort is still negligible compared to the actual computation of the moments, since for each occupied voxel we will have to compute

$$\sum_{n=0}^{n_{max}} \sum_{r=0}^{n} \sum_{s=0}^{n-r} 1 = \frac{1}{6}n_{max}^3 + n_{max}^2 + \frac{11}{6}n_{max} + 1 \tag{24}$$

integrals and aggregate the results across all the voxel grid. Therefore, we will implement this next part of the computation on the GPU.

Conceptually, finding each geometric moment M_{rst} is as simple as looking up the result of each of the 1D integrals for every occupied voxel, multiplying them together, then sum the results from all voxels. However, some care must be exercised when aggregating the results from the voxels. Floating-point addition is not associative, especially when very dissimilar terms are to be summed, and it is obvious that the results of equation 23 can be many orders of magnitude apart if evaluated on a voxel close to the origin or if on a voxel close to $\|\boldsymbol{x}\| = 1$, especially for moments of high order.

Before calling the actual computation kernel, we transfer to the device memory the lookup table and the voxel data, then allocate and initialize to zero an array sized $N \times n_{max} \times n_{max} \times n_{max}$ to hold partial results of the geometric moments. We configure a block size $n_{max} \times n_{max} \times 1$ and a grid size $N \times 1$. Inside the computation kernel, `threadIdx` maps to constant s and t per thread and `blockIdx` maps to a constant j per block. We will call this kernel once per k, so that index is also constant for every thread in every block.

Using this configuration, all the threads in one block will work on the same whole single line of voxels, each of them iterating on every possible values for r, and all of them will also need the 1D integrals on x for those values of r, so it makes sense to load these data into shared memory, which greatly improves performance. To do this we employ the cooperative reading technique, which consists in each of the k threads in each block reading the positions $\bmod(k)$ of the data source corresponding to their index number.

	$N = 200$			$N = 180$			$N = 80$		
	CPU	GPU	Speed-up	CPU	GPU	Speed-up	CPU	GPU	Speed-up
$n_{max} = 20$	1809 ms	58 ms	31×	1293 ms	43 ms	30×	313 ms	11 ms	28×
$n_{max} = 15$	1188 ms	45 ms	26×	897 ms	34 ms	26×	117 ms	8 ms	15×
$n_{max} = 10$	857 ms	36 ms	24×	718 ms	27 ms	27×	70 ms	6 ms	12×
$n_{max} = 5$	764 ms	30 ms	25×	661 ms	23 ms	29×	59 ms	5 ms	12×

Table 1. Comparison of computation times of the Zernike moments between sequential (CPU) and parallel (GPU) implementations of the algorithm.

In order to contain the numerical error, the threads iterate on the voxels of the line, that correspond to variations on the x coordinate, from the voxel that corresponds most closely to $x = 0$ outwards. Since all other parameters are fixed, this guarantees that the terms we add to the sum are in increasing order. Finally, each thread writes on the positions $j \times (0 \dots n_{max}) \times s \times t$ of the results array, so there are no two threads either in the same or in different blocks writing on the same position.

As we told before, this kernel is called once per k, and again the iteration on k is done from the position that corresponds most closely to $z = 0$ outwards. After the iteration is completed, each of the $j \times \dots \times \dots \times \dots$ subarrays of the results contains the contributions of planes of constant j, and a new kernel adds them down, one thread per (r, s, t) tuple, from the centre outwards.

Interestingly, we learned that the cost of accessing global memory, albeit to load data on shared memory, impacts the performance of the algorithm severely, so we modified the data representation to use just one bit per voxel, packing them on the x dimension into `unsigned short` cells of 16 voxels each. Obviously this packing operation only makes sense on the CPU. Since we have to iterate on the whole dataset to pack it, the computation of the centroid comes essentially for free, so we stuck to the CPU implementation for that.

3.3 Zernike moments

Once the geometric moments are computed, we only need to make linear combinations of them as indicated in equation 17. For that purpose we use another kernel on the GPU, each thread being responsible for a particular (n, l, m) combination. At the start of the execution of the program we load into the global memory of the GPU the whole table of χ^{rst}_{nlm} coefficients, and now we cannot share them between different threads, but the volume of data to read is very small, so it does not impact performance.

Again, we need to exercise caution to add each of the terms of the sum: the χ^{rst}_{nlm} coefficients range from 10^0 to 10^9 for $n_{max} = 20$ and the geometric moments also have a great dynamic range. Double precision floating-point numbers can only hold a mantissa of ~ 16 decimal digits and now we have no means to order the terms of the sum beforehand, so we use the Kahan summation algorithm[14] to double the precision of the mantissa, obtaining results within 8 decimal places of the true value, computed symbollically with Maple.

4. RESULTS

We have performed tests both using a sequential implementation of the algorithm, all in the CPU, and the parallel implementation on GPU we have described. The test rig is a Core i7 960 clocked at 3.2 GHz, equipped with 12 GB DDR3 RAM and a nVidia GTX 465. All three datasets represent one person in different poses in a cubic space measuring $2\,\mathrm{m} \times 2\,\mathrm{m} \times 2\,\mathrm{m}$, discretized at different resolutions.

The measurements of computation time can be seen on table 1. The biggest gains are found on high orders and big grid sizes because some costs are expected to be constant, for example runtime overhead, and some other are more dependent on the size of the grid than on the maximum order.

5. CONCLUSIONS

We have shown a parallelisation of a known algorithm for computing 3D Zernike moments that makes them practical for use in real-time interactive applications. Just by carefully arranging the operations and without making drammatical

changes to the original algorithm, computation time is reduced well over one order of magnitude while retaining numerical accuracy and stability.

The main limitation of this implementation is the same than in the original sequential algorithm: we can only compute moments up to the order we have precalculated the complex coefficients which are necessary to combine geometrical moments. In order to compute them online if unavailable for the required order, very high precision integer arithmetic is required, but some efforts are being made to develop multiprecision libraries on the GPU, which would open a future way to improve the implementation.

Acknowledgments

This work has been partially supported by Telefónica I+D and the Spanish Administration agency CDTI under project CENIT-VISION 2007-1007 and by the Ministerio de Ciencia e Innovación of the Spanish Government under project TEC2007-67764 (SmartVision). We also want to thank especially Patrick Min for writing (and very kindly modifying at our request) Binvox,[15] a mesh voxelizer based on parity count and ray stabbing[16] that we have employed to generate our test datasets.

REFERENCES

[1] M. Hu, "Visual pattern recognition by moment invariants," *Information Theory, IEEE Transactions on* **8**(2), pp. 179–187, 1962.

[2] M. Teague, "Image analysis via the general theory of moments," *JOSA* **70**(8), pp. 920–930, 1980.

[3] A. Khotanzad and Y. Hong, "Invariant Image Recognition by Zernike Moments," *IEEE Transactions on Pattern Analysis and Machine Intelligence* **12**(5), pp. 489–497, 1990.

[4] B. Li and J. Shen, "Fast computation of moment invariants," *Pattern Recognition* **24**(8), pp. 807–813, 1991.

[5] R. Mukundan and K. Ramakrishnan, "Fast computation of Legendre and Zernike moments," *Pattern Recognition* **28**(9), pp. 1433–1442, 1995.

[6] C. Chong, P. Raveendran, and R. Mukundan, "A comparative analysis of algorithms for fast computation of Zernike moments," *Pattern Recognition* **36**(3), pp. 731–742, 2003.

[7] M. Ujaldon, "GPU acceleration of Zernike moments for large-scale images," in *Proceedings of the 2009 IEEE International Symposium on Parallel & Distributed Processing*, pp. 1–8, IEEE Computer Society, (Washington, DC, USA), 2009.

[8] N. Canterakis, "3D Zernike moments and Zernike affine invariants for 3D image analysis and recognition," in *11th Scandinavian Conf. on Image Analysis*, pp. 85–93, 1999.

[9] M. Kazhdan, T. Funkhouser, and S. Rusinkiewicz, "Rotation invariant spherical harmonic representation of 3D shape descriptors," in *Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pp. 156–164, Eurographics Association, 2003.

[10] J. Tangelder and R. Veltkamp, "A survey of content based 3D shape retrieval methods," *Multimedia Tools and Applications* **39**(3), pp. 441–471, 2008.

[11] M. Novotni and R. Klein, "Shape retrieval using 3D Zernike descriptors," *Computer-Aided Design* **36**(11), pp. 1047–1062, 2004.

[12] J. Pozo, M. Villa-Uriol, and A. Frangi, "Efficient 3D Geometric and Zernike Moments Computation from Unstructured Surface Meshes," *Pattern Analysis and Machine Intelligence, IEEE Transactions on* **PP**(99), p. 1, 2010.

[13] P. Bose, A. Maheshwari, and P. Morin, "Fast approximations for sums of distances, clustering and the Fermat-Weber problem," *Computational Geometry* **24**(3), pp. 135–146, 2003.

[14] W. Kahan, "Pracniques: further remarks on reducing truncation errors," *Communications of the ACM* **8**, pp. 40–, January 1965.

[15] P. Min, "Binvox, a 3d mesh voxelizer," `http://www.cs.princeton.edu/~min/binvox/`.

[16] F. Nooruddin and G. Turk, "Simplification and Repair of Polygonal Models Using Volumetric Techniques," *IEEE Transactions on Visualization and Computer Graphics* **9**(2), pp. 191–205, 2003.