Accurate Moment Computation Using the GPU

Adarsh Krishnamurthy*, Sara McMains
University of California, Berkeley
Berkeley, CA, USA

ABSTRACT

We present algorithms for computing accurate moments of solid models that are represented using multiple trimmed NURBS surfaces. Our algorithms make use of programmable Graphics Processing Units (GPUs) to accelerate the computations. We evaluate the surface coordinates and normals accurately, with theoretical bounds, using our GPU NURBS evaluator. We have developed a framework that makes use of this data to evaluate surface integrals of trimmed NURBS surfaces in real time. With our framework, we can compute volume and moments of solid models with theoretical guarantees. The framework also supports local geometry changes, which is useful for providing interactive feedback to the designer while the solid model is being designed. We can compute the center of mass and check for stability of the solid model interactively. Applications of such real-time moment computation include deformation modeling, animation, and physically based simulations.

Categories and Subject Descriptors

I.3.3 [**Computer Graphics**]: Hardware ArchitectureGraphics Processors;; I.3.5 [**Computer Graphics**]: Computational Geometry and Object ModelingGeometric Algorithms, Languages, and Systems; G.1.4 [**Numerical Analysis**]: Quadrature and Numerical DifferentiationGaussian Quadrature

Keywords

Geometric Algorithms, Moments, Volume, NURBS, GPU, Hybrid CPU/GPU Algorithms

1. INTRODUCTION

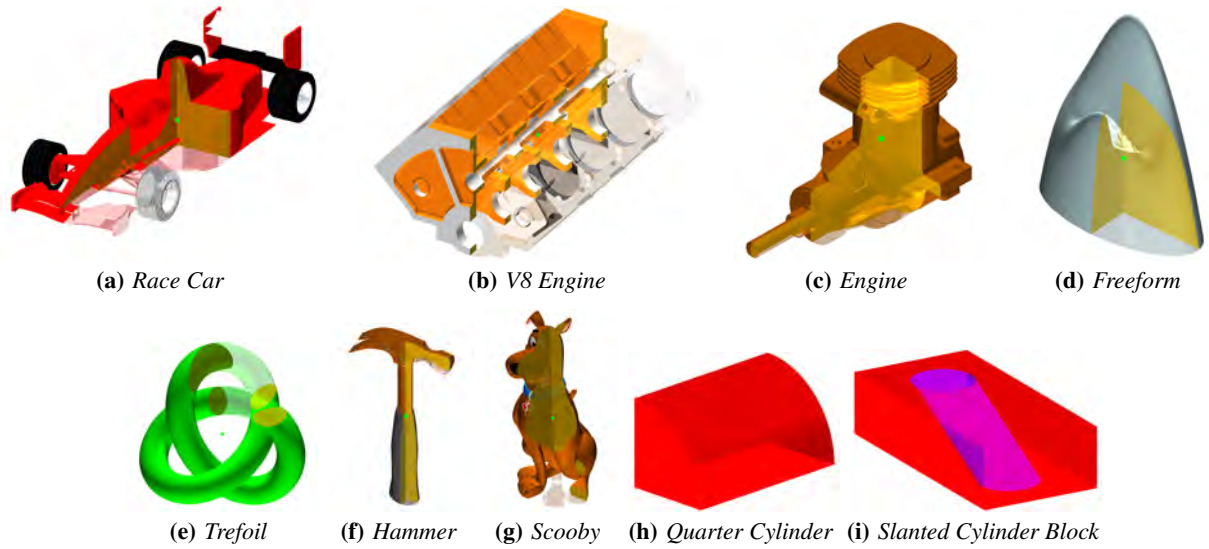
Geometric moments of solid bodies are intrinsic properties of their underlying shape that can provide important design cues to aid in Computer-aided Design (CAD). Computing moments is also essential for physically based simulations that help in improving realism in animations. In addition, many Computer-Aided Manufacturing (CAM) analyses depend on computing accurate volume, center of mass, and moments of inertia. For example, a part that is fixtured using its center of mass will prevent any unbalanced loads on the fixturing system. The moments of inertia can be used to compute the loads that might be transferred to the fixturing system. Although designers have an intuitive sense of the location of the center of mass and principle moments of inertia, accurate feedback about these properties can be a valuable asset while designing. The zeroth-order moment measures the volume enclosed by the solid body. Computing accurate volumes is essential since it helps the designer estimate the amount of raw material that might be required to manufacture a particular part. This is essential especially in the case of molding, where the volume computation is essential to maintain quality while manufacturing. If a mold is filled incompletely, it leads to voids in the final part; conversely, filling a mold with excess material leads to flash that needs to be trimmed later. Thus, accurate interactive feedback about moments can aid the designers in spotting any inconsistencies and make corrections early in the design process, keeping costs low.

Computing accurate moments of modern CAD models interactively is not straightforward due to the presence of curved freeform surfaces. Trimmed Non-Uniform Rational B-Spline (NURBS) surfaces are the de facto representation for curved surfaces in modern CAD systems. Moments of objects made of such freeform surfaces are currently being computed by commercial solid modeling software by first evaluating and tessellating the surfaces and then computing the moments of the tessellated object by projecting them to a common plane that passes through the middle of the object [28]. This approach, in addition to being extremely slow and computationally intensive, is dependent on the tessellation resolution for the accuracy of the solution; the surface has to be very finely tessellated to get the required accuracy. Recent advances in programmable GPUs provide an alternative to accelerate the computations. We have developed a GPU-accelerated algorithm to compute moments of solid objects made of trimmed NURBS surfaces in real time. These algorithms exploit the parallelism of the GPU to provide immediate visual feedback to the designer. The algorithm updates the moment values interactively while the designer changes the design of the part.

In our method, we calculate moments by first converting the volume integrals for moments to surface integrals using the divergence theorem. We then use the GPU to compute these surface integrals. Thus, our algorithm is not restricted to computing moments but can be used to compute general case surface integrals of NURBS surfaces. Computing surface integrals of free-form surfaces form an important part of several physical simulation algorithms, including finite element analysis. In such cases, our algorithm provides a framework to compute surface integrals of NURBS surfaces rapidly and accurately. We have developed different GPU-based numerical integration techniques that can be used to evaluate surface integrals of NURBS.

*e-mail:{adarsh|mcmains}@me.berkeley.edu

2010 ACM Symposium of Solid and Physical Modeling (SPM '10), Haifa, Israel



(a) *Race Car* **(b)** *V8 Engine* **(c)** *Engine* **(d)** *Freeform*

(e) *Trefoil* **(f)** *Hammer* **(g)** *Scooby* **(h)** *Quarter Cylinder* **(i)** *Slanted Cylinder Block*

Figure 1: Computing the volume and center of mass of complex objects made of multiple trimmed-NURBS surfaces. The center of mass is marked with a green dot; the objects are rendered partially transparent.

In this paper, we provide a hybrid CPU/GPU algorithm that can be used to compute moments. Our main contributions include:

- A GPU accelerated moment computation algorithm that can be used to calculate the volume, center of mass, moment of inertia, etc. of solid models that are represented using multiple trimmed NURBS surfaces. Our computations are independent of the tessellation of the solid models. They allow for interactive updates of moments while the model is being edited.
- A GPU algorithm for performing numerical surface integration on trimmed NURBS surfaces for moment computations. We provide details of using the GPU to perform midpoint, 2-point Gaussian quadrature, or 3-point Gaussian quadrature integration schemes. We compare the accuracy of our algorithm to theoretical results.
- Error estimates for our moment computations. They allow for user-defined tolerance values that are essential for integrating our algorithms in a CAD system.

2. RELATED AND PREVIOUS WORK

Volume and geometric moment computations have been fundamental to many geometric applications and the literature covering them is vast. We provide a brief summary of some key papers that are close to our work in Section 2.1. We then provide a brief summary of our previous GPU NURBS contributions in Section 2.2. We make use of some of the concepts that we outline in this section in our current work.

2.1 Related Work

Properties such as volume, moment of inertia, etc. can be classified as integral properties that are defined by triple (volumetric) integrals over subsets of three-dimensional Euclidean space. Initial work on computing volume of curved objects by approximating them with polyhedral facets was performed by Messner et al. [21]. However, the errors introduced due to the approximation were not analyzed. Lee and Requicha published a two-part paper that discussed the methods that were known up to that time for computing the integral properties of solids. In their first paper [19], they summarize the methods for computing volume and moments of solid bodies represented using Boundary Representation (B-rep), primitive instancing, etc. In their subsequent paper [20], they focus on Constructive Solid Geometry (CSG) and propose a new algorithm based on cellular approximation using octrees. They also predict the accuracy of their algorithm in approximating the integral properties.

Boundary Representations (b-reps) are the de facto standard for representation of solid models in current CAD systems. Lee and Requicha classified the different algorithms for computing geometric moments of b-reps as either direct integration methods or divergence theorem methods. For polyhedral models, direct integration methods can be easily applied since the integration is performed over planes. Cattani and Paoluzzi [3] developed a finite integration method for the computation of various-order monomial integrals over polyhedral solids and surfaces. This method can be used for exact evaluation of inertial properties of homogeneous polyhedral objects. Mirtich [22] developed a fast algorithm to compute moments of polyhedral objects by successively applying the divergence theorem. Timmer and Stern [29] developed a computational scheme that operates directly on parametric bi-cubic spline patches to compute the inertial properties of solids. Their method relied on computing the intersection curves of adjacent surfaces of the solid accurately to compute the integral properties. For interactive applications, Ochoa et al. [5] developed a method to compute moments of piecewise polynomial surfaces that is based on the divergence theorem. They outline several applications in animation where the models can be solved to match the moments. However, their method relies on modeling the objects using a particular type of cubic spline surfaces known as C^1-surface splines to maintain interactivity. For graphical animation applications that use subdivision surfaces, Peters and Nasri [23] developed a method to compute the volume of a solid enclosed by subdivision surfaces by estimating the volume of the local convex hull near extraordinary points. Soldea et al. [27] developed an analytical method for computing moments of free-form objects that use either parametric surfaces or a constant set of trivariate functions as boundary. In their method, they compute integrals of b-spline blending functions to compute the moments of free-form objects.

GPU geometric algorithms are currently gaining popularity due to the performance gains achievable by using GPUs. However, algorithms to compute geometric moments were limited to volume computations since many of them calculated the volume by rendering to the screen. Kim et al. [11] use the depth buffer while rendering to the screen to compute the volume of general shapes and use it for buoyancy simulation. Khardekar et al. [10, 9] have developed a GPU algorithm to compute the undercut volumes in molds that can then be used to choose an optimal parting direction while using multi-part molds. GPUs were also used for solving standard geometric problems such as surface-surface intersections [2] and collision detection [6, 12, 14, 7]. However, there has been limited use of GPUs to compute integral properties for 3-dimensional solid bodies since algorithms that render to the screen are limited to image-space precision.

One of the advantages of developing a GPU algorithm that uses the divergence theorem is that the integration method can also be used for the evaluation of surface integrals of polynomial forms. These forms commonly arise in physical simulations that use Finite Element Analysis (FEA). Scalar valued functionals, such as energy functionals, are usually evaluated as surface integrals; they can be used to measure the quality of surfaces [13]. Recently, researchers at the University of Wisconsin have developed a method of analyzing 3-D beams by performing surface integration over the boundary of the beam [26, 8].

On analyzing the related work in the field, we find that even though numerous methods for computing moments and surface integrals exist, there has been only limited analysis on the accuracy of the computations. In addition, there has been limited work on parallelizing the computations to improve their performance. We will try to address both these issues in this paper.

2.2 Previous Work

Our moment computation algorithms build on our previous papers on GPU NURBS evaluation and modeling. We present a short outline of our GPU algorithms that were explained in detail in [15, 17]. In our NURBS evaluation paper, we developed a method to evaluate a mesh of points on a NURBS surface directly using the GPU. Our algorithm used a fragment program to evaluate a NURBS surface of arbitrary degree in several passes. After evaluation we have samples on the NURBS surface as 4-component vectors—(x, y, z, w) coordinates—in space stored as a texture on the GPU. While rendering, we interpret these values stored in the texture as vertex coordinates using a Vertex Buffer Object (VBO) and display the mesh directly on the screen. However, this previous work used uniform parametric grid spacing in our evaluations for display and modeling operations; in this paper, we adapt this to perform evaluations with non-uniform grid spacing.

Using the NURBS evaluator, we were able to build a GPU framework to perform geometric operations such as surface-surface intersections, sketching [16, 17], silhouette curve evaluation, and minimum distance computations [18]. These geometric algorithms were still based on uniform parametric evaluation of NURBS surfaces and made use of surface Axis-Aligned Bounding-Boxes (AABBs) to accelerate the computations.

In our NURBS modeling work, in order to bound the errors, we find the maximum possible deviation K of a curved surface from the linearized approximation. The analytical expression for the deviation based on the surface curvature is given by Filip et al. [4]. They show that if a parametric C^2 surface is evaluated at $(n+1) \times (m+1)$ grid of points, the deviation of the surface from the piecewise linear approximation cannot exceed the constant K defined by Equations (1) – (4).

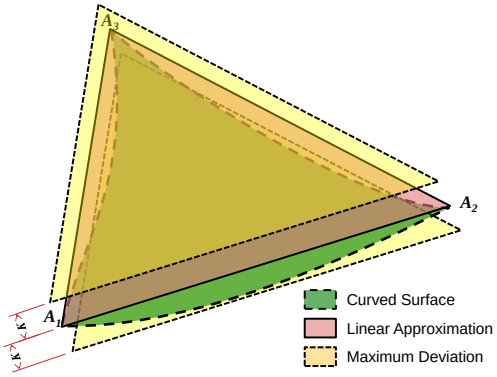


Figure 2: The maximum possible deviation of the actual surface from a linear approximation is K. In this case, the surface is bounded by two parallel triangles that are at a distance K from the linear approximation.

$$K_1 = \max_{\forall(u,v)} \left[\max \left(\left| \frac{\partial^2 x}{\partial u^2} \right|, \left| \frac{\partial^2 y}{\partial u^2} \right|, \left| \frac{\partial^2 z}{\partial u^2} \right| \right) \right] \tag{1}$$

$$K_2 = \max_{\forall(u,v)} \left[\max \left(\left| \frac{\partial^2 x}{\partial u \partial v} \right|, \left| \frac{\partial^2 y}{\partial u \partial v} \right|, \left| \frac{\partial^2 z}{\partial u \partial v} \right| \right) \right] \tag{2}$$

$$K_3 = \max_{\forall(u,v)} \left[\max \left(\left| \frac{\partial^2 x}{\partial v^2} \right|, \left| \frac{\partial^2 y}{\partial v^2} \right|, \left| \frac{\partial^2 z}{\partial v^2} \right| \right) \right] \tag{3}$$

$$K = \frac{1}{8} \left(\frac{1}{n^2} K_1 + \frac{2}{nm} K_2 + \frac{1}{m^2} K_3 \right) \tag{4}$$

Given any three nearby points evaluated on the surface using the uniform grid of size $n \times m$, we can approximate the surface linearly using the triangle formed using these points (Figure 2). We can also bound the coordinates of the curved surface with two parallel triangles that are at a distance K from the linear approximation, since the maximum possible deviation of the surface is K.

3. MATHEMATICAL FORMULATION

In this section, we briefly explain the mathematical formulations required for moment computations; for a more detailed explanation please refer to [25]. We make use of Gauss's divergence theorem to convert volume integrals to an integral over the boundary surface of the volume. The divergence theorem is a special case of the generalized n-dimensional Stokes' theorem that is restricted to three dimensions.

THEOREM 1. ***(Divergence Theorem)*** *Given a vector field* $\mathbf{f}$ *defined over a closed bounded region,* $V \subset \Re^3$*, whose boundary is a piecewise smooth orientable surface S, the volume integral of the divergence of* $\mathbf{f}$ *over* V *equals the surface integral of the normal component of* $\mathbf{f}$ *over* S*.*

$$\nabla \cdot \mathbf{f} = \sum \frac{\partial f_i}{\partial x_i} \tag{5}$$

$$\int_V \nabla \cdot \mathbf{f} \, dV = \int_S \mathbf{f} \cdot \hat{\mathbf{n}} \, dS \tag{6}$$

Equations (6) formalizes the statement of the divergence theorem. However, as noted by the theorem statement, the theorem is applicable only if both the vector field $\mathbf{f}$ and the region V satisfy

some basic conditions. The vector field must be continuous and have continuous first partial derivatives in the region containing V. In addition, V itself should be closed and its boundary surfaces must be orientable and piecewise continuous. The piecewise continuous surface condition expands the applicability of the theorem to many practical 3-dimensional objects, since it overcomes the limitation of having undefined normals at sharp edges of objects. The divergence theorem is applicable to any 3-dimensional object as long as it is 2-manifold. Using the divergence theorem, we can convert volume integrals, which are difficult to evaluate for complex solid objects made of multiple trimmed NURBS surfaces, to surface integrals that can be easily evaluated over the NURBS surfaces.

The evaluations of surface integrals require the calculation of surface normals. We will briefly present the equations for evaluating NURBS derivatives and normals in Section 3.1; please refer to [24, 17] for more details.

3.1 Evaluation of NURBS Normals

Recall that the parameterized NURBS surface can be represented as a 3-component vector (Equation (7)) which is evaluated as the 4-component vector shown in Equations (8). The N_i^ps and N_j^qs are the B-spline basis functions of degree p and q respectively; $(x_{ij}, y_{ij}, z_{ij}, w_{ij})$s are the NURBS control points defined as a quad mesh.

$$\mathbf{S}(u,v) = \frac{\mathbf{X}}{w}, \mathbf{X} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{7}$$

$$\begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = \begin{pmatrix} \sum_{i=0}^{n}\sum_{j=0}^{m} N_i^p(u)N_j^q(v)x_{ij} \\ \sum_{i=0}^{n}\sum_{j=0}^{m} N_i^p(u)N_j^q(v)y_{ij} \\ \sum_{i=0}^{n}\sum_{j=0}^{m} N_i^p(u)N_j^q(v)z_{ij} \\ \sum_{i=0}^{n}\sum_{j=0}^{m} N_i^p(u)N_j^q(v)w_{ij} \end{pmatrix} \tag{8}$$

Given a NURBS surface, we can evaluate the partial derivatives and normals; Equation (9) gives the partial u-derivative for the NURBS surface that is evaluated from the derivatives of the basis function of degree p with respect to u, represented as $N_{i,u}^p(u)$. The partial derivative of the surface with respect to v can also be evaluated in a similar manner. In this work, we assume all the weights (w) are positive and hence no poles can occur in S or its partial derivatives. Finally, the normal to the surface is evaluated as the cross-product of the u and v partial dervatives (Equation (11)).

$$\mathbf{S}_{,u}(u,v) = \frac{\mathbf{X}_{,u}w - \mathbf{X}w_{,u}}{w^2} \tag{9}$$

$$\begin{pmatrix} x_{,u} \\ y_{,u} \\ z_{,u} \\ w_{,u} \end{pmatrix} = \begin{pmatrix} \sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,u}^p(u)N_j^q(v)x_{ij} \\ \sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,u}^p(u)N_j^q(v)y_{ij} \\ \sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,u}^p(u)N_j^q(v)z_{ij} \\ \sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,u}^p(u)N_j^q(v)w_{ij} \end{pmatrix} \tag{10}$$

$$\mathbf{n}(u,v) = \mathbf{S}_{,u}(u,v) \times \mathbf{S}_{,v}(u,v) \tag{11}$$

3.2 Surface Integrals of Parametric Surfaces

Surface integrals of parametric surfaces are straightforward to compute due to the presence of an underlying 2-dimensional parameterization. We can convert the surface integrals to integrals over the parametric domain by changing the variables. In Equation (12), P represents the parametric (u, v) domain and J is the Jacobian for the transformation. It can be shown that the Jacobian can be computed to be numerically equal to the length of the normal of the parametric surface (Equation (13)).

$$\int_S dS = \int_P |J| \, dP \tag{12}$$

$$|J| = |\mathbf{n}| \tag{13}$$

The volume integrals simplify with the application of the divergence theorem as given by Equation (14). In particular, for NURBS surfaces, the parametric domain is a square domain with the (u, v) range $[0, 1] \times [0, 1]$ and dP can be replaced with the product $du\, dv$.

$$\begin{aligned} \int_V \nabla \cdot \mathbf{f} \, dV &= \int_S \mathbf{f} \cdot \hat{\mathbf{n}} \, dS \\ &= \int_P \mathbf{f} \cdot \hat{\mathbf{n}} \, |\mathbf{n}| \, dP \\ &= \int_P \mathbf{f} \cdot \mathbf{n} \, dP \\ &= \int_P \mathbf{f} \cdot \mathbf{n} \, du \, dv \end{aligned} \tag{14}$$

3.3 Moments of Solid Bodies

By choosing appropriate vector functions for $\mathbf{f}$, we can compute the moments of solid bodies with uniform densities. By applying the divergence theorem to the solid body, we can compute the moments by computing the contribution of each surface and sum the results. In Equation (15), P_i represents the parametric surfaces that make up the solid body.

$$\int_V \nabla \cdot \mathbf{f} \, dV = \sum_i \int_{P_i} \mathbf{f} \cdot \mathbf{n} \, du \, dv \tag{15}$$

By setting $\nabla \cdot \mathbf{f} = 1$, we get the zeroth-order moment, M_0, the volume of the solid body. However, there are many different choices for $\mathbf{f}$ that satisfy this condition. One option is to choose $\mathbf{f}$ as shown in Equation (16).

$$M_0 = \sum_i \int_{P_i} \begin{pmatrix} 0 \\ 0 \\ z \end{pmatrix} \cdot \mathbf{n} \, du \, dv = \sum_i \int_{P_i} z \, n_z \, du \, dv \tag{16}$$

The first-order moments, defined by Equations (17)–(20), can also be computed by carefully choosing $\mathbf{f}$. For example, in order to compute the first-order moment M_x, we need to set $\nabla \cdot \mathbf{f} = x$. Similar to the volume computation case, there are several choices for $\mathbf{f}$ that satisfy this requirement; we choose the x and y components to be both 0, and the function xz for the z component. The other first-order moments can be computed in a similar manner. The center of mass C_M of the object is computed by dividing the first-order moments by the volume of the object (Equation (21)).

$$\mathbf{M_1} = \begin{pmatrix} M_x \\ M_y \\ M_z \end{pmatrix} = \begin{pmatrix} \int_V x\,dV \\ \int_V y\,dV \\ \int_V z\,dV \end{pmatrix} \tag{17}$$

$$M_x = \sum_i \int_{P_i} x\,z\,n_z\,du\,dv \tag{18}$$

$$M_y = \sum_i \int_{P_i} y\,z\,n_z\,du\,dv \tag{19}$$

$$M_z = \sum_i \int_{P_i} \left(\frac{z^2}{2}\right) n_z\,du\,dv \tag{20}$$

$$\mathbf{C_m} = \begin{pmatrix} C_x \\ C_y \\ C_z \end{pmatrix} = \begin{pmatrix} M_x/M_0 \\ M_y/M_0 \\ M_z/M_0 \end{pmatrix} \tag{21}$$

The second-order moments form the components of the inertia tensor, I, given by Equation (22). The components of the inertia tensor can be computed using Equations (23)–(28).

$$I = \begin{pmatrix} M_{yy}+M_{zz} & -M_{xy} & -M_{xz} \\ -M_{xy} & M_{xx}+M_{zz} & -M_{yz} \\ -M_{xz} & -M_{yz} & M_{xx}+M_{yy} \end{pmatrix}$$

$$= \begin{pmatrix} \int_V (y^2+z^2)\,dV & -\int_V xy\,dV & -\int_V xz\,dV \\ -\int_V xy\,dV & \int_V (x^2+z^2)\,dV & -\int_V yz\,dV \\ -\int_V xz\,dV & -\int_V yz\,dV & \int_V (x^2+y^2)\,dV \end{pmatrix} \tag{22}$$

$$M_{xx} = \sum_i \int_{P_i} x^2\,z\,n_z\,du\,dv \tag{23}$$

$$M_{yy} = \sum_i \int_{P_i} y^2\,z\,n_z\,du\,dv \tag{24}$$

$$M_{zz} = \sum_i \int_{P_i} \left(\frac{z^3}{3}\right) n_z\,du\,dv \tag{25}$$

$$M_{xy} = \sum_i \int_{P_i} x\,y\,z\,n_z\,du\,dv \tag{26}$$

$$M_{yz} = \sum_i \int_{P_i} y \left(\frac{z^2}{2}\right) n_z\,du\,dv \tag{27}$$

$$M_{xz} = \sum_i \int_{P_i} x \left(\frac{z^2}{2}\right) n_z\,du\,dv \tag{28}$$

4. MOMENT COMPUTATION

We first give a broad overview of our algorithm that makes use of the theoretical formulation explained in Section 3. Given a solid object that is made of multiple trimmed NURBS surfaces, we compute the total moment by summing the moment contribution from each surface. If the surface is a flat plane, we directly compute its moment contribution by using the triangulation of the plane. If it is a NURBS or a trimmed NURBS surface, we compute its moment contribution by performing surface integration using our GPU algorithm. It must be noted that the object must be 2-manifold for the computed moments to be valid. However, the main advantage of computing the moments from surface integrals is that the algorithm is robust in handling the small gaps between trimmed surfaces that exist in a tolerant solid model, since we do not evaluate points on the edges. Tolerant solid modeling provides a method for handling these small gaps; any two points that lie within a user-defined tolerance is taken to represent the same point. This is essential to create a watertight object when the model has many trimmed-NURBS surfaces that vary in parameterization along their common edges.

In our algorithm, we divide each NURBS surface into sub-patches equal to the number of knot-intervals in each parametric direction; we call this the base number of sub-patches. Based on the number of sub-patches and the type of integration scheme used, we create a vector of u and v parametric positions where we want to evaluate the surface. If it is a trimmed NURBS surface, we also generate a trim-texture [15] based on the number of sub-patches. We then compute the moment contribution from each patch by multiplying the corresponding functions for moments with the integration weights. Finally, we compute the sum of all the sub-patch moment contributions to get the moment contribution of the surface.

4.1 GPU Implementation

Once we have evaluated the surface coordinates and normals at the integration points using our GPU NURBS evaluator, we compute the moment contribution of each surface sub-patch in parallel using the GPU. We compute four moment values simultaneously, since GPUs are optimized for simultaneously computing values using the four RGBA channels. For example, we compute the volume and the three first moments simultaneously in a single pass. We compute the moment contribution for each sub-patch by multiplying the moment functions with the integration weights. The moment functions are polynomial functions of the surface coordinates that correspond to the moment being computed; they are given by the corresponding equations in Section 3.3. The integration weights are based on the type of integration used (Section 5).

Once we have computed the individual contributions of each the sub-patches, we sum the values to get the surface moments by using GPU reduction. This operation is one of the fundamental operations used for GPU CAD in our GPU framework [18]. GPU reductions include reducing the given input to a single value such as computing the sum, min, max, etc. If the given input is a square texture with a size that is a power of two, then we reduce four adjacent values (the sum in this case) to a single value in a given pass. Thus the total reduction operation can be performed in $O(\log n)$ passes and hence, it is very efficient. On the other hand, if the input is not a square texture, then we perform the reduction in three stages. In the first stage, we reduce only two values along the height or the width direction until we reach a power-of-two texture. In the second stage, we reduce along the larger dimension until we reach a square texture. Finally, we perform the normal reduction for square power-of-two textures in the third stage.

We make use of Cg shader programs to implement the GPU operations to perform the moment computations and reductions. This

is because we found in our preliminary testing that our shader programs provide better performance for NURBS evaluations compared to our optimized CUDA implementation of the same algorithm. In addition, making use of shader programs to perform these operations makes our implementation cross-platform; our implementation can run on both NVIDIA and AMD GPUs.

5. NUMERICAL SURFACE INTEGRATION

We compute the surface integrals of NURBS surfaces numerically using the Gaussian quadrature rule. In numerical analysis, a quadrature rule approximates the definite integral of a function using a weighted sum of function values at specified points within the domain of integration. An n-point Gaussian quadrature rule yields an exact result for the integration of polynomials up to degree $2n-1$ with suitable choice of points t_i and weights w_i (Equation (29)). The domain of integration for such a rule is conventionally taken as $[-1, 1]$; however, the domain can be easily changed to any value by using a dummy variable for integration (Equation (30)).

$$\int_{-1}^{1} f(t)\,dt \approx \sum_{i=1}^{n} w_i f(t_i) \tag{29}$$

$$\int_{a}^{b} f(t)\,dt \approx \frac{b-a}{2} \sum_{i=1}^{n} w_i f\left(\frac{b-a}{2} t_i + \frac{b+a}{2}\right) \tag{30}$$

We can extend the quadrature rules to compute 2-dimensional integrals by having two weights; one for each direction. The integration rule can then be modified as given by Equation (31). As in the 1-dimensional case, the domain of integration is $[-1, 1] \times [-1, 1]$; this can be converted to any rectangular domain by changing the integration variables.

$$\int_{-1}^{1} f(t)\,dt \approx \sum_{i=1}^{n} \sum_{j=1}^{n} w_i\, w_j\, f(t_{ij}) \tag{31}$$

In the following sections, we provide details of the weights and evaluation points for performing 1-point, 2-point, and 3-point Gaussian quadrature integration of NURBS surfaces.

5.1 1-point Gaussian Quadrature Scheme

The 1-point quadrature or the mid-point scheme is the simplest of the numerical integration schemes. It approximates the integral value with a single point that is evaluated at the center of the integration domain. The standard weight used for the point is 4.0 for the 2-dimensional case (Equation (32)). This being the simplest rule for integration, it can only integrate accurately up to degree 1 or linear polynomials.

$$\int_{-1}^{1} f(t)\,dt \approx 4\, f(0, 0) \tag{32}$$

For integrating over the surface of a NURBS patch, we divide the parametric domain into sub-patches along the u and v parametric directions. For each sub-patch, we evaluate the function at its mid-point; we perform the required change of variable implicitly. The mid-point scheme is not accurate enough in computing the surface integrals in many practical cases; however, it can be used to get a rough approximation for the solution since it is easy to evaluate. Another important advantage is that the mid-point scheme can be implemented on the GPU using uniform grid spacing; the evaluation points are separated uniformly along the u and v directions in the parametric domain.

5.2 2-point Gaussian Quadrature Scheme

Most solid models that are created using popular CAD systems such as SolidWorks are usually composed of bi-cubic NURBS surfaces. In order to compute the volume accurately for such solid models, we need to use the 2-point quadrature scheme. Implementing the 2-point quadrature scheme is slightly more involved than the mid-point scheme since the spacing between the evaluation points is not uniform. However, the weights used are constant and they are equal to 1.0; the sum of the weights of the four evaluation points is 4.0. The evaluation points in the normalized $[-1, 1]$ domain are $-1/\sqrt{3}$ and $1/\sqrt{3}$. Equation (33) expands the integration terms for the 2-dimensional case.

$$\begin{aligned}\int_{-1}^{1} f(t)\,dt \approx\; & f\left(\frac{-1}{\sqrt{3}}, \frac{-1}{\sqrt{3}}\right) + f\left(\frac{-1}{\sqrt{3}}, \frac{1}{\sqrt{3}}\right) \\ & + f\left(\frac{1}{\sqrt{3}}, \frac{-1}{\sqrt{3}}\right) + f\left(\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}\right)\end{aligned} \tag{33}$$

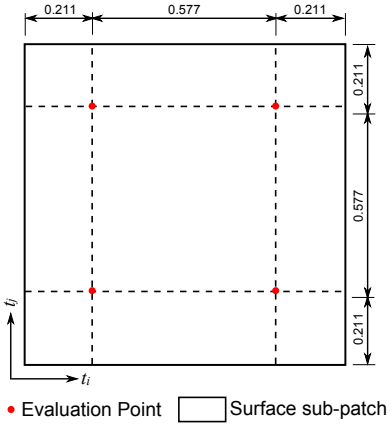


Figure 3: Distribution of evaluation points in the 2-point Gaussian quadrature integration scheme. All the four integration points have a uniform weight of 1.0.

Figure 3 gives the fraction of the intervals between the evaluation points in a single sub-patch of the NURBS surface. The evaluation points are positioned symmetrically with respect to both the u and v directions inside the sub-patch. Figure 4 gives an example of the position of the evaluation points in a complete NURBS surface.

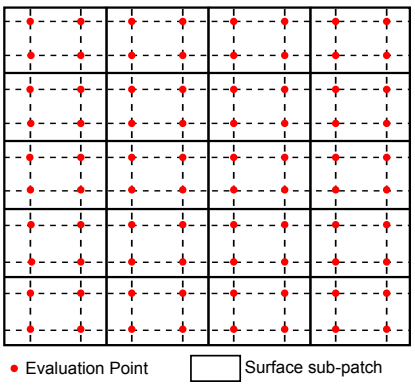


Figure 4: Example of the evaluation points' location in the parametric domain for computing the surface integrals using the 2-point Gaussian quadrature integration scheme.

In this example, the NURBS surface is divided into 4×5 sub-patches in the u and v directions respectively. The marked points' parametric positions are sent as a vector to the GPU evaluator and are then used to evaluate the surface coordinates and normals.

5.3 3-point Gaussian Quadrature Scheme

Computing the higher order moments accurately in a solid model consisting of bi-cubic NURBS patches requires the 3-point quadrature scheme. Computing using the 3-point scheme requires the calculation of different weights for the different evaluation points. The parametric positions, t_is, of the evaluation points and their corresponding weights, w_is, are given by Equation (35). The definite integral in the canonical domain is approximated by the double-sum of the product of the weights and the integrand evaluated at the corresponding points. It can be noted that there are nine evaluation points in the 2-dimensional case; the sum of the product of the weights is 4.0.

$$\int_{-1}^{1} f(t)\,dt \approx \sum_{i=1}^{3}\sum_{j=1}^{3} w_i\, w_j\, f(t_i, t_j) \tag{34}$$

$$\begin{aligned} t_1 &= -\sqrt{\left(\frac{3}{5}\right)},\ w_1 = \frac{5}{9} \\ t_2 &= 0,\ w_2 = \frac{8}{9} \\ t_3 &= \sqrt{\left(\frac{3}{5}\right)},\ w_3 = \frac{5}{9} \end{aligned} \tag{35}$$

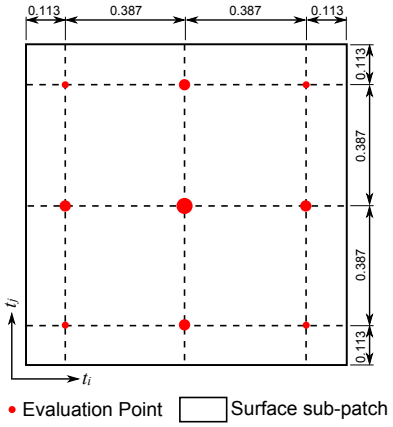


Figure 5: Distribution of evaluation points in the 3-point Gaussian quadrature integration scheme. The size of the integration point indicates its relative weight; the sum of all the weights equals 4.0.

Figure 5 gives the location of the 9 evaluation points as a fraction of the sub-patch size. The size of the evaluation point indicates its relative weight. The center point has the maximum weight followed by the 4 points near the edge midpoints; the 4 corner points have the least weight.

5.4 Surface Integrals of Trimmed NURBS

We compute the surface integrals of trimmed NURBS in two stages. In the first stage, we treat them as un-trimmed surfaces and compute the moment contribution of each sub-patch using the methods explained in Section 5. We classify the sub-patches into three different cases: inside, outside or on the boundary of the trim curves. If the sub-patch lies inside a trimmed region, its contribution to the total surface integral is taken to be 0. Similarly, if the sub-patch lies outside the trimmed region, its contribution to the surface integral is taken as 1. The contribution for all the sub-patches that lie on the boundary of the trim-curves is weighted by the fraction of the sub-patch that lies outside the trimmed region.

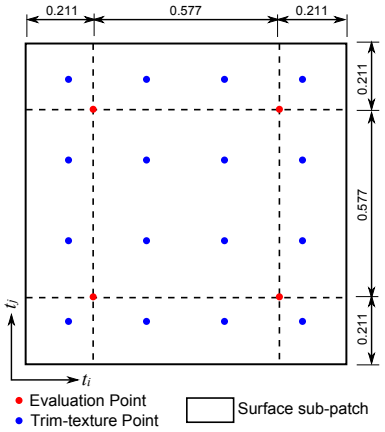
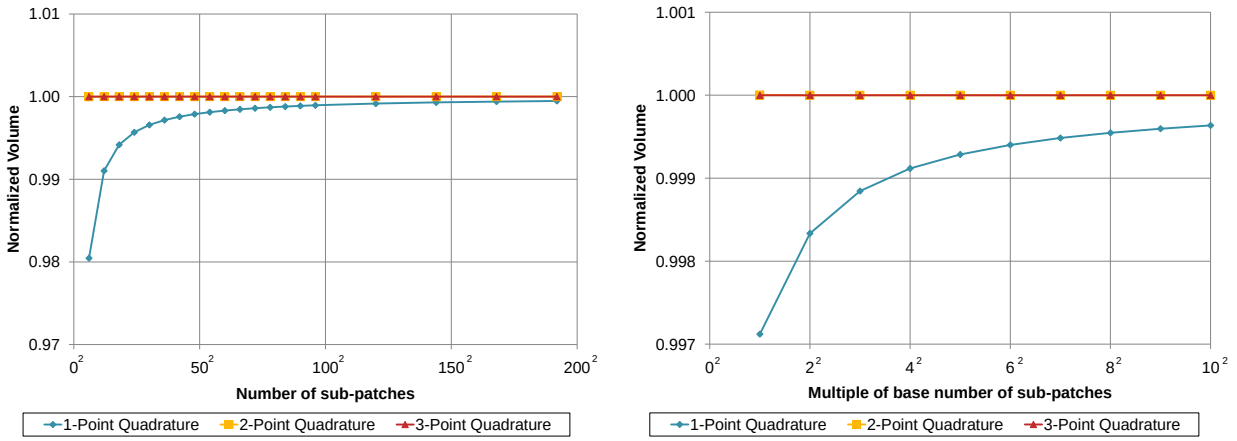


Figure 6: Example of the distribution of the trim-texture points for a case with 4 points in each parametric direction. The 2-point Gaussian quadrature integration scheme evaluation points are also shown for comparison.

In order to perform the weighting operation, we generate a trim-texture on the GPU by using a method we developed in our previous work [15]. In this method, we create a binary texture that has the value 0 in the trimmed regions of the surface in the parametric domain. In order to obtain an accurate value for the moments, we set the height and width of the trim texture to be three or four times the number of sub-patches in the u and v direction. We find the fraction of the number of points that lie outside the trimmed region in a given sub-patch and multiply the moment contribution of the sub-patch with this fraction. Figure 6 shows an example where the trim texture is evaluated 4 times more densely than the number of sub-patches in each of the u and v parametric direction. This leads to a total of 16 positions inside the sub-patch where the trim is evaluated. The moment contribution of the patch is multiplied with the fraction of the number of points that lie outside the trimmed region.

6. ERROR ANALYSIS

In this section, we derive estimates for the error in computing the volume of a solid body; this analysis can be extended in a similar manner to compute the error in higher-order moments. Computing the errors in a Gaussian integration scheme directly is difficult since it involves computing the $2n$-order derivative for an n-point integration scheme. A known simpler method is to compute the error as the difference between two different Gaussian integration schemes. It can be shown that this is a good estimate for the error since the integration value converges to the theoretical value as the order of Gaussian integration is increased. However, the surface integrals have to be evaluated twice using different orders of integration in this method. In our case, we estimate the error in the integration by computing the bounds for the coordinates used for the integration; the bounds are compued from the maximum possible deviation of the surface from a linear approximation. Using this method, we compute the error in the volume of several practical CAD models in Section 7.



(a) *Quarter Cylinder* **(b)** *Slanted Cylinder Block*

Figure 7: Graphs showing the accuracy of our volume computation algorithm for two cases whose volume can be theoretically computed. The volume is normalized with respect to the theoretical volume. The x-axis for the right graph shows the number of sub-patches used for integration of each NURBS surface as a multiple of their base number of sub-patches. It can be noted that the 2 and 3-point quadrature schemes give exact results with very few sub-patches.

In our method, for each surface sub-patch, we find the second partial derivatives of the surface as explained in Section 2.2. We then compute the maximum deviation K for each surface sub-patch based on the maximum second partial derivative. This gives the maximum deviation in the coordinates used for the surface integration. By neglecting higher order error terms, we can show that the error in volume computation can be calculated approximately using Equation (36). Intuitively, this error estimate measures the maximum possible deviation of the volume from the polyhedral approximation of the NURBS surface. It can be shown from Figure 2 that this error estimate measures the volume between the two parallel planes that are located at a distance K from the linear approximation.

$$\begin{aligned} \Delta M_0 &\approx \sum_i \int_{P_i} |\, \Delta z\, n_z \,|\, du\, dv \\ &\leq \sum_i \int_{P_i} |\, 2\, K\, n_z \,|\, du\, dv \end{aligned} \tag{36}$$

We calculate this error simultaneously while computing the moment values. For each sub-patch in the surface, we compute the value of K and store it in a separate texture. While computing the moments, we compute the error terms for each sub-patch. We then perform the multiplication and reduction operations similar to the moment computations explained in Section 4.1. Finally, after summing all the error contributions from each sub-patch, we get the total error for the surface. We perform the same operations on all the surfaces to get the total error in the volume of the solid object. In case of trimmed-NURBS surfaces, we consider only the error contributions from those sub-patches that lie outside the trimmed region.

7. RESULTS

We timed our GPU-accelerated queries on a 2.40GHz CPU (dual core) running Windows XP with 3GB of RAM and an NVIDIA GeForce 9600M GT GPU with 256MB graphics memory. First, we discuss the accuracy of our algorithm by using it to compute moments of trimmed-NURBS solid models whose volume can be calculated theoretically. Next, we vary our integration schemes and the number of sub-patches used for the integration and check for the convergence of the solution. Finally, we compare the actual values of volume and center of mass of complex CAD models with the moment values obtained using ACIS; we also compare the time for the computations.

Accuracy of the Integration

Since the exact value for the moments are very difficult to obtain for practical models, we tested the accuracy of our integration on a single NURBS patch of a quarter cylinder. We chose a cylinder since we can accurately compute its theoretical volume and center of mass for comparison, yet it is non-trivial for our GPU algorithm to evaluate since it is a rational surface. In addition, the cylinder was placed horizontally in the coordinate system to make sure the moment contributions from the cylindrical surfaces are not zero. As expected, the higher order quadrature schemes provide a more accurate answer, particularly with smaller numbers of sub-patches (Figure 7(a)).

In order to assess the accuracy of our algorithm with the presence of trimmed NURBS surfaces, we constructed a solid object consisting of a block with a slanted cylindrical section cut from it (Figure 1(i)). We can theoretically calculate the volume and center of mass of this object since the slanted cylinder is still a prism with a circular base. Figure 7(b) shows the accuracy as a function of the base number of patches used for evaluating each NURBS surface. We set the base number of sub-patches equal to the number of knot intervals in the respective u and v parametric direction of the NURBS surface, the total base number of sub-patches for this model being 128.

Volume and Error Analysis of CAD Objects

In order to test the applicability of our algorithm to realistic solid models, we computed the volume of several CAD models with multiple trimmed NURBS surfaces. For comparison, we calculated the volume of the models using ACIS with accuracy 0.001 measured as a fraction of the computed moment; calculating to a higher accuracy using ACIS took too long to be practically applicable. In addition, the ACIS algorithm does not have a bound on the precision of the mass properties because of hard-coded convergence criteria in their functions [28]. If the accuracy does not meet the requested

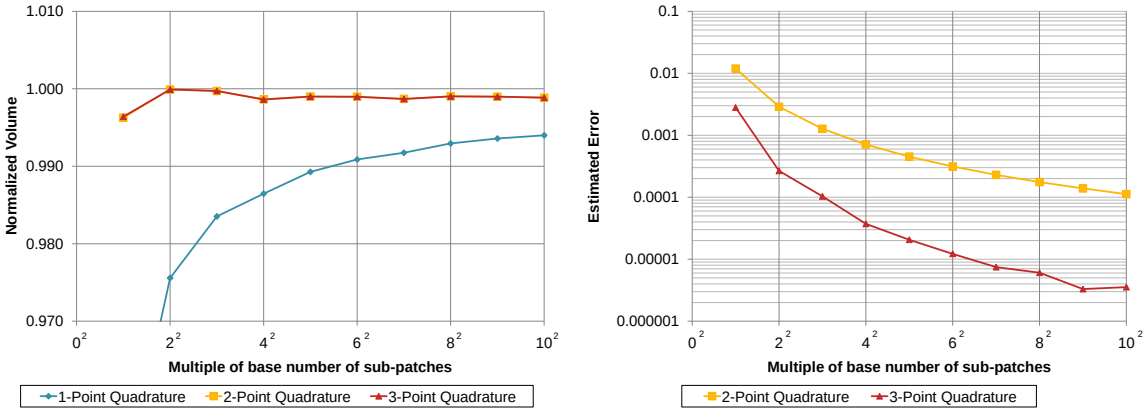


Figure 8: Graphs showing the convergence of our volume computation algorithm for the "Hammer" model. The volume is shown as a fraction of the volume computed by ACIS. The x-axis shows the number of sub-patches used for integration as a multiple of the base number of sub-patches of the model, the base number of sub-patches being 3859 in this case. Note logarithmic scale in the y-axis of the error graph.

value, they tighten the convergence criteria and repeat the calculation. However, if the mass properties remain unchanged, they are as close as can be achieved by their algorithm. In our algorithm, we estimate the error in the volume while using 2-point or 3-point quadrature integration as explained in Section 6.

We computed the volume of these objects using 1-point, 2-point, and 3-point quadrature by varying the number of sub-patches that are used for the integration of each NURBS surface. As before, we set the base number of sub-patches equal to the number of knot intervals in the respective u and v parametric direction of the NURBS surface. Figure 7 shows the convergence in the volume computation for the "Hammer" model shown in Figure 1(f).

Moment Computation Results

We compared the results for evaluating the volume and center of mass of different CAD objects that were made of multiple trimmed-NURBS surfaces using both our GPU algorithm and ACIS. Table 1 summarizes the results of the computations. However, it should be noted that neither the ACIS mass properties function nor our GPU algorithm have been optimized for performance. In addition, the time taken by our GPU algorithm includes the time for evaluating all the NURBS surfaces, which takes the largest percentage of the total time ($\approx 90\%$). The values were computed using the 2-point quadrature scheme with the number of sub-patches equal to twice the base number of sub-patches for each surface in both parametric directions. It can be noted that our GPU algorithm computes accurate moments with low estimated errors.

An advantage of our GPU algorithm is that the updates to the volume and center of mass due to changes in a single surface can be performed interactively. For example, the GPU algorithm computes the volume and the center of mass for the "Freeform" model in less than 0.02s. This means that even though the initial moment computation for complex models takes more than a second, the moment values can be updated up to 50 times per second while interactively editing the solid model.

8. CONCLUSIONS

We have developed a hybrid framework that uses GPUs to accelerate moment computations. Our moment computations can be performed interactively, while the model is being edited. Our algorithms have error estimates and they are based on object-space resolution instead of just image-space resolution. They make use of actual surface data and not just the tessellation, which make them independent of tessellation errors. We also show tremendous performance and accuracy improvements over existing commercial CPU-based systems.

Our GPU-based surface integration algorithms can be extended for use in analysis tools such as FEA. Accurate and fast surface integration will aid in interactive analysis of complex objects that will provide functional feedback to the designer. Such functional feedback will reduce the design lead time of a component, ultimately resulting in significant cost savings.

Acknowledgments

We would like to thank NVIDIA and AMD for providing us with their hardware. The models used in the paper were downloaded from 3D Content Central [1]. We would also like to thank the reviewers for their valuable comments and suggestions. This material is based upon work supported in part by SolidWorks Corporation, UC Discovery under Grant No. DIG07-10224, and the National Science Foundation under CAREER Award No. 0547675.

References

[1] 3D Content Central. http://www.3dcontentcentral.com, 2009.

[2] S. Briseid, T. Dokken, T. R. Hagen, and J. O. Nygaard. *Computational Science - Lecture Notes in Computer Science*, volume 3994/2006, chapter Spline Surface Intersections Optimized for GPUs, pages 204–211. Springer, 2006.

[3] C. Cattani and A. Paoluzzi. Boundary integration over linear polyhedra. *Computer Aided Design*, 22(2):130–135, 1990.

[4] D. Filip, R. Magedson, and R. Markot. Surface algorithms using bounds on derivatives. *Computer Aided Geometric Design*, 3(4):295–311, 1987.

[5] C. Gonzalez-Ochoa, S. McCammon, and J. Peters. Computing moments of objects enclosed by piecewise polynomial surfaces. *ACM Transactions on Graphics*, 17(3):143–157, 1998.

[6] N. K. Govindaraju, S. Redon, M. C. Lin, and D. Manocha. CULLIDE: Interactive collision detection between complex models in large environments using graphics hardware. In *ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, pages 25–32. Eurographics Association, 2003.

[7] A. Greß, M. Guthe, and R. Klein. GPU-based collision detection for deformable parameterized surfaces. *Computer Graphics Forum*, 25(3):497–506, 2006.

Object	GPU				ACIS		
	Volume	Estimated Volume Error Fraction	Center of Mass	Time (s)	Volume	Center of Mass	Time(s)
Scooby	1.99516×10^{8}	6.248×10^{-2}	0.438 654.325 92.532	1.516	1.99601×10^{8}	0.330 653.821 92.381	15.547
Trefoil	8.57171×10^{6}	4.658×10^{-3}	-0.225 0.251 0.112	0.063	8.57174×10^{6}	-0.225 0.251 0.112	0.175
Freeform	9.79816×10^{5}	1.251×10^{-3}	-0.002 57.366 -2.883	0.015	9.82948×10^{5}	0.001 57.433 -2.876	0.718
V8 Engine	9.97957×10^{6}	1.933×10^{-6}	-1.011 93.136 261.283	1.188	9.99160×10^{6}	-0.626 93.535 260.435	5.078
Engine	1.29932×10^{5}	1.000×10^{-5}	-0.741 20.112 12.640	0.235	1.30284×10^{5}	-0.748 20.003 12.726	0.922
Race Car	1.83031×10^{9}	5.534×10^{-5}	-1.005 356.936 1362.790	0.703	1.83799×10^{9}	0.021 356.846 1364.280	4.140

Table 1: Moment and center of mass values computed by our GPU algorithm and ACIS for different CAD models. Our values were computed using the 2-point quadrature scheme with 2^2 times the base number of sub-patches for each surface. The errors were estimated using our method explained in Section 6.

[8] K. Jorabchi, J. Danczyk, and K. Suresh. Efficient and automated analysis of potentially slender structures. *Journal of Computing and Information Science in Engineering*, 9(4), 2009.

[9] R. Khardekar. *Real-time manufacturability feedback*. PhD thesis, University of California, Berkeley, Mechanical Engineering Department, 2008.

[10] R. Khardekar and S. McMains. Fast layered manufacturing support volume computation on GPUs. In *Proceedings of the ASME Design Engineering Technical Conferences*. ASME, 2006.

[11] J. Kim, S. Kim, H. Ko, and D. Terzopoulos. Fast GPU computation of the mass properties of a general shape and its application to buoyancy simulation. *Visual Computer*, 22(9):856–864, 2006.

[12] P. Kipfer, M. Segal, and R. Westermann. UberFlow: a GPU-based particle engine. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, pages 115–122. ACM, 2004.

[13] L. Kobbelt. Robust and efficient evaluation of functionals on parametric surfaces. In *SCG '97: Proceedings of the thirteenth annual symposium on computational geometry*, pages 376–378. ACM, 1997.

[14] A. Kolb, L. Latta, and C. Rezk-Salama. Hardware-based simulation and collision detection for large particle systems. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, pages 123–131. ACM, 2004.

[15] A. Krishnamurthy, R. Khardekar, and S. McMains. Optimized GPU evaluation of arbitrary degree NURBS curves and surfaces. *Computer Aided Design*, 41(12):971–980, 2009.

[16] A. Krishnamurthy, R. Khardekar, S. McMains, K. Haller, and G. Elber. Performing efficient NURBS modeling operations on the GPU. In *ACM Symposium on Solid and Physical Modeling*, pages 257–268. ACM, 2008.

[17] A. Krishnamurthy, R. Khardekar, S. McMains, K. Haller, and G. Elber. Performing efficient NURBS modeling operations on the GPU. *IEEE Transactions on Visualization and Computer Graphics*, 15(4):530–543, 2009.

[18] A. Krishnamurthy, S. McMains, and K. Haller. Accelerating geometric queries using the GPU. In *SIAM/ACM Joint Conference on Geometric and Physical Modeling*, pages 199–210. ACM, 2009.

[19] Y. T. Lee and A. A. G. Requicha. Algorithms for computing the volume and other integral properties of solids. I. Known methods and open issues. *Communications of the ACM*, 25(9):635–641, 1982.

[20] Y. T. Lee and A. A. G. Requicha. Algorithms for computing the volume and other integral properties of solids. II. A family of algorithms based on representation conversion and cellular approximation. *Communications of the ACM*, 25(9):642–650, 1982.

[21] A. M. Messner and G. Q. Taylor. Algorithm 550: Solid polyhedron measures [z]. *ACM Transactions on Mathematical Software*, 6(1):121–130, 1980.

[22] B. Mirtich. Fast and accurate computation of polyhedral mass properties. *Journal of Graphics Tools*, 1(2):31–50, 1996.

[23] J. Peters and A. Nasri. Computing volumes of solids enclosed by recursive subdivision surfaces. *Computer Graphics Forum*, 16:89–94, 1997.

[24] L. A. Piegl and W. Tiller. *The NURBS Book*. Springer, second edition, 1997.

[25] W. Rudin. *Principles of Mathematical Analysis*, chapter Integration of Differential Forms, pages 253–275. McGraw-Hill, 3 edition, 1976.

[26] W. A. Samad and K. Suresh. CAD-integrated analysis of 3-D beams: A surface-integration approach. *Engineering with Computers*, Submitted Aug 2009.

[27] O. Soldea, G. Elber, and E. Rivlin. Exact and efficient computation of moments of free-form surface and trivariate based geometry. *Computer-Aided Design*, 34(7):529–539, 2002.

[28] Spatial Corporation. *ACIS Geometric Modeler: User Guide*, 2009. Version 20.0.

[29] H. Timmer and J. Stern. Computation of global geometric properties of solid objects. *Computer-Aided Design*, 12(6):301–304, 1980.