Parametric Deformation of Discrete Geometry for Aerodynamic Shape Design

George R. Anderson *
Stanford University, Stanford, CA 94305

Michael J. Aftosmis †
NASA Ames Research Center, Moffett Field, CA 94305

Marian Nemec ‡
Science and Technology Corp., Moffett Field, CA 94305

We present a versatile discrete geometry manipulation platform for aerospace vehicle shape optimization. The platform is based on the geometry kernel of an open-source modeling tool called Blender and offers access to four parametric deformation techniques: lattice, cage-based, skeletal, and direct manipulation. Custom deformation methods are implemented as plugins, and the kernel is controlled through a scripting interface. Surface sensitivities are provided to support gradient-based optimization. The platform architecture allows the use of geometry pipelines, where multiple modelers are used in sequence, enabling manipulation difficult or impossible to achieve with a constructive modeler or deformer alone. We implement an intuitive custom deformation method in which a set of surface points serve as the design variables and user-specified constraints are intrinsically satisfied. We test our geometry platform on several design examples using an aerodynamic design framework based on Cartesian grids. We examine inverse airfoil design and shape matching and perform lift-constrained drag minimization on an airfoil with thickness constraints. A transport wing-fuselage integration problem demonstrates the approach in 3D. In a final example, our platform is pipelined with a constructive modeler to parabolically sweep a wingtip while applying a 1-G loading deformation across the wingspan. This work is an important first step towards the larger goal of leveraging the investment of the graphics industry to improve the state-of-the-art in aerospace geometry tools.

I. Introduction

GEOMETRY is the beginning and the end of aerodynamic design. Yet aerospace geometry tools lag behind the state of the art in geometry processing. The aerospace design community tends to develop in-house, discrete geometry tools that focus on rapidly providing application-specific functionality while sacrificing generality, flexibility and often user-friendliness. Meanwhile, the computer graphics (CG) industry has for decades invested in geometry manipulation techniques, far surpassing the scattershot development of niche aerospace geometry tools.

In this work we explore a strategy of leveraging advanced CG geometry tools and customizing them for aerospace design. These tools offer a wealth of sophisticated deformation methods and surface processing routines. Their robust and efficient geometry algorithms can be used either interactively with intuitive graphical user interfaces (GUIs), or via scripting interfaces on the back-end, allowing them to serve as automated geometry engines in design environments. Furthermore, CG geometry tools are highly extensible. The various special deformation techniques developed by the aerospace community in recent years can be incorporated as plugins with modest effort. New deformation techniques can be rapidly prototyped on a flexible and unified geometry manipulation platform.

Our goal is to develop a general platform for deformation of discrete aerospace geometries. The platform is based on a powerful, open-source 3D modeling suite called Blender.[1] Several deformation techniques

*Ph.D. Candidate, Department of Aeronautics and Astronautics; george.anderson@stanford.edu.
†Aerospace Engineer, Advanced Supercomputing Division, MS 258-5; michael.aftosmis@nasa.gov. Associate Fellow AIAA.
‡Senior Research Scientist, Advanced Supercomputing Division, MS 258-5; marian.nemec@nasa.gov. Member AIAA.

are natively supported, including lattice deformation, cage-based deformation, and skeletal deformation. Deformation can be interactively parameterized using custom panels in Blender's GUI. Custom deformation methods for aerospace design can be developed using a scripting API. We extend Blender's feature set by implementing constraint-based (direct manipulation) deformation as a plugin using this API. After a geometry has been parameterized, our platform serves as a fully-automated geometry engine, providing on-demand deformations, along with surface sensitivities for gradient-based shape optimization. The platform can be used alone or pipelined with other geometry tools, including constructive modelers, to enhance an existing design environment.

To provide perspective on our work, we begin with a comparison of the constructive and deformation geometry modeling paradigms, followed by a focused survey of discrete geometry deformation techniques. The main thrust begins in section IV, where we introduce Blender and discuss its geometry manipulation capabilities. Thereafter we present our geometry platform, which customizes Blender for aerospace design. We conclude with several shape optimization examples to concretely assess performance of the new tool.

II. Geometry Modeling Paradigms

Every geometric modeler is either a constructor or a deformer. In *constructive* modeling, a geometry is built from scratch according to a sequential recipe. Familiar constructive modeling elements include B-splines, Bernstein polynomials, Bézier curves, NURBS surfaces, and class/shape functions, all of which are used in aerospace design.[2–6] CAD modeling tools are often favored for aerospace design, because they draw many of these common constructive elements into a single package.[7] Another class of constructive modeling is subdivision surface modeling, in which a coarse initial mesh is evolved to an arbitrarily fine mesh by iterative refinement.[8] Subdivision surfaces have been used for turbine blade optimization.[9]

Constructive modeling is prevalent in aerospace design, because it is well-suited both to the beginning and to the end of the design process. Its intuitiveness and topological flexibility are advantageous during conceptual design. CAD is the standard format for manufacturing once design is completed. Constructive models also provide a continuous master geometry source, from which discipline-specific surrogate models can be derived for structural, environmental or aerodynamic analysis.

The fundamental weakness of constructive modeling is the geometry recipe itself, which dictates not only what a surface can do, but what it cannot do. Poor geometry parameterization can cripple shape optimization by blindly and inadvertently restricting the design space. This is a persistent problem in CAD-based design environments. A CAD operator cannot possibly foresee all the parameters that will be necessary for shape optimization. Worse, a designer may not have access to the original constructive model or the expertise necessary to alter it. A discrete geometry, such as a surface triangulation or patched surface grid, may be the only format available. Such *legacy* geometry is beyond the scope of constructive modeling, unless the surface is painstakingly (and approximately) replicated in a constructive modeler. Finally, the modeling software itself may be unavailable, expensive, unreliable, or even defunct. A designer frustrated with the difficulty of maintaining a fragile constructive modeling pipeline may well choose to discard it in favor of discrete geometry. Alternately, constructive modeling pipelines can be enhanced by adding deformation techniques to the mix.

Deformation is an entirely distinct modeling paradigm. Deformation techniques encode modifications to a baseline geometry. Whereas constructive modelers are the correct tools for establishing topology during conceptual design, deformation methods are more appropriate for detailed shape optimization. Highly flexible reshaping is possible with deformation techniques, freeing the designer from the constraints of the original constructive parameterization. The design space is opened up, enabling shape optimization tools to discover optimal shapes that the original constructive parameterization could not reach. Some deformation methods can even deform a computational flow mesh in tandem with the surface, a valuable capability when using body-fitted approaches.

Optimized geometries typically must be converted back to CAD for manufacturing. While in the past this commonly-raised point was a serious criticism of discrete geometry, recent developments are removing some of the barriers between constructive modeling and freeform deformation. Commercial tools now exist that can transfer geometries between the two domains.[10,11] Ideally, future choices between constructive modeling and deformation will be made based on their respective strengths and not on mundane issues of format conversion.

III. Deformation of Discrete Geometry

The modern aerospace designer has a smorgasbord of deformation techniques to choose from. But wading through the literature on the vast body of methods can be daunting. We provide here a classification of the major techniques to give context to our work. Further discussion can be found in a 1999 survey by Samareh.[12]

Deformation methods are either surface-based or volumetric. The important distinction between the two is the geometric region they focus on. Surface-based methods focus on the surface, while having no awareness of the encompassing volume. Conversely, volumetric methods focus on the surrounding space, while being agnostic to the details of the embedded surface. This sharp dichotomy results in methods with complementary strengths and weaknesses, which we discuss in the following sections. Figure 1 depicts the various techniques we discuss.

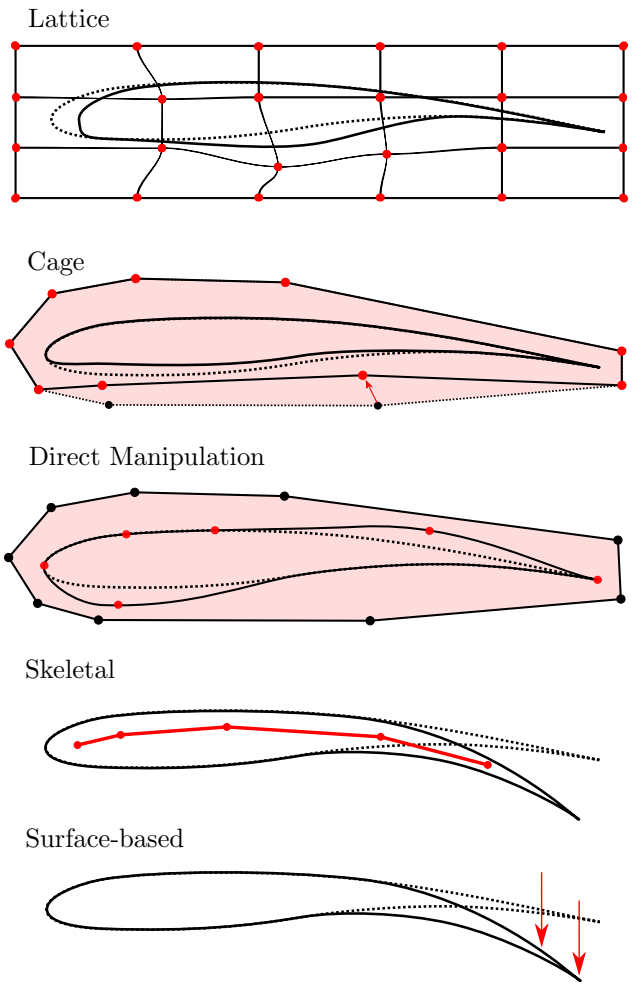


Figure 1: Deformation techniques, depicted schematically. User-adjusted parameters are in red. Dashed line is the initial airfoil, solid line is the deformed airfoil. Though illustrated here in 2D, each technique extends straightforwardly to 3D.

A. Surface-Based Deformation

Surface-based techniques define a deformation function on the surface itself. The simplest forms, such as bump functions, use explicit analytic deformation functions. As they are conceptually straightforward, analytic surface-based techniques have a long history of use in aerodynamic design.[13,14]

Many modern surface-based deformation techniques instead use variational (or energy-based) formulations, which imitate the deformation physics of real surfaces. Specifically, they compute the shape that minimizes some global energy norm, which can be tailored to produce shapes that minimize curvature, surface area, or other surface characteristics. Berkenstock and Aftosmis[15] adapted a nonlinear variational method[16] for use in aerodynamic design. The deformations were of high quality, but the method proved too expensive for fast-paced design. Linear variational methods achieve much faster deformations by introducing simplifications to the physics, at the cost of sacrificing some deformation quality. Botsch and Sorkine[17] provide a concise review of linear variational deformation methods.

Surface-based methods are intrinsically surface-aware, which allows for physically realistic deformations and makes it easier to preserve geometric features, such as straight leading edges on wings. However, their reliance on global mesh information means that their computational cost scales with the complexity of the surface discretization. Variational surface-based methods are usually unaffordable for finely detailed aerospace configurations, where the number of vertices can easily exceed one million. One common strategy for evading the computational scaling issue is to work instead with a coarse surrogate of the original surface, such as the control net of a subdivision surface. Because the surrogate is much simpler, it can be deformed rapidly. A fine surface can then be generated from the coarse mesh by fast iterative subdivision. Dubé et al. used this technique for turbine blade design.[9]

Surface-based methods also impose exacting requirements on the geometry format. Typically a closed and manifold triangulation is required. As non-aerodynamic disciplines use various incompatible geometry formats (e.g. structural elements), surface-based methods are ill-suited for multi-disciplinary optimization. Furthermore, the quality of surface-based deformation is limited by the quality of the surface discretization.

A poor surface can cripple surface-based methods. This forces the designer to worry about surface mesh quality, an esoteric topic unto itself. Lastly, surface-based methods cannot deform volumetric structures like CFD meshes, and so they must rely on separate specialized mesh deformation codes.

B. Volumetric Deformation

Volumetric (or spatial) methods define a deformation function within a volume. Surface meshes are then embedded in this volume. Conceptually speaking, the volume is deformed and the embedded surfaces passively follow. Volumetric methods have three key advantages. First, their computational cost is nearly independent of the embedded surface's complexity.[a] Second, unlike surface-based methods, volumetric methods are impartial to the discrete geometry format. They can simultaneously deform surface triangulations, flow meshes, and even structural models. And third, the deformation quality of volumetric methods is independent of the surface discretization quality. Disadvantages stem from the inherent lack of surface-awareness. Volumetric deformation techniques may have difficulty preserving geometric features and if not properly constrained can easily produce illogical shapes. In section C we discuss two intuitive and reliable methods for properly constraining volumetric deformers to produce expected deformations.

In the following subsections we give two examples of standard volumetric techniques. These examples are by no means exhaustive. Gain and Bechmann provide a more comprehensive, user-oriented comparison of volumetric deformation techniques.[18] In another notable method that we do not examine here, radial basis functions are used to perform unified surface deformation and CFD mesh deformation.[19] This method has recently been applied to the design of transonic wings and helicopter blades.[20–22]

1. *Free-Form Deformation*

The oldest volumetric deformation technique (and still the most common) is free-form deformation (FFD), introduced in 1986 by Sederberg and Parry.[23] Figure 2 shows an example of FFD. A 3D lattice of control points establishes an analytic function on the interior of the lattice. The user moves the control points, which warp the enclosed volume, and any geometry inside the volume is deformed accordingly. FFD methods use polynomial splines with compact support that localize the deformation while ensuring continuity. FFD methods are increasingly being used in aerospace design,[2,24–26] and mature commercial tools exist.[11]

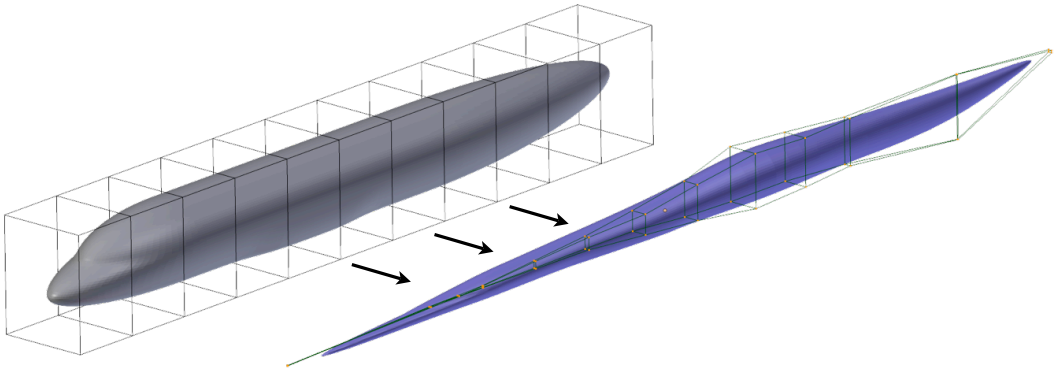


Figure 2: A lattice reshapes a transport fuselage into a notional supersonic fuselage.

2. *Cage-based Deformation*

An alternate volumetric deformation technique encloses the geometry in a coarse closed surface mesh, called the *cage*, as depicted in Figure 3. The basic idea is to use a coarse mesh to guide the deformation of a complex surface underneath. Like in lattice deformation, the design variables are control points, but in cage-based methods they are located only on the surface of the cage, not throughout the volume of the lattice. The cage can take any shape, allowing the designer to specify the number of design variables and to cluster design variables for detailed control in critical regions.

[a] Strictly speaking, no deformation algorithm can scale better than linearly with the number of surface mesh vertices. However, variational surface-based methods usually scale far worse, as they involve solving matrix systems of dimension equal to the number of vertices.

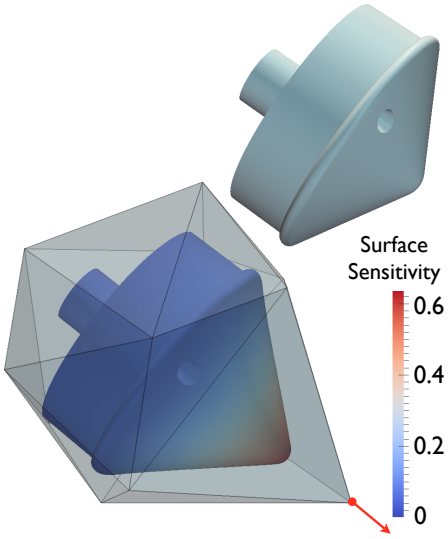


Figure 3: A cage reshapes the nose of a wind tunnel model of a reentry capsule. Top: original model. Bottom: deformed model. Surface colors indicate the surface sensitivities to the highlighted control point.

The deformation function in cage-based methods is the solution of a smooth partial differential equation (PDE) on the interior of the cage. This PDE solution is used to determine bindings between the cage control points and the geometry vertices, a process called *embedding*. While the PDE solution may be somewhat computationally intensive (seconds to minutes depending on the volume discretization), the PDE solution and embedding happen only once, in a pre-computation step. After the geometry is embedded, deformations are extremely rapid, involving only matrix-vector products.

Cage-based methods are somewhat more intuitive than FFD techniques because they conform more closely to the surface. They also involve a more manageable number of control points and give the designer more flexibility in designing a parameterization. Cage-based methods have proliferated in recent years in the CG industry,[18,27,28] but they have yet to be widely used for aerospace design. A notable exception is the work of Anderson et al.,[29] in which a cage-based method was used for compressor blade design.

C. Reducing the Design Space with Intuitive Parameters

Free-form and cage-based deformation methods are flexible and general techniques. They can effortlessly parameterize any discrete geometry, and their underlying mathematics ensure smooth and rapid deformations. Unfortunately, the clouds of control points involved are cumbersome to use as design variables. For example, a $10 \times 10 \times 10$ lattice already involves 3000 design parameters, as each control point can move in three orthogonal directions. In fast-paced design, a smaller set of physically intuitive design variables is preferred. Ideally, the user would be able to rely on a deformation method's good intrinsic properties (such as smoothness), while being shielded from the minutia of its underlying mathematics and from the excessive number of its control parameters.

The most obvious design space reduction strategy is to group control points by proximity. Manual grouping can be used to achieve a specific result, but this level of micromanagement can be avoided by using more automated and visually intuitive methods.

1. *Skeletal Deformation*

One appealing technique is to embed a "skeleton" in the geometry, as shown in Figure 4. Each "bone" in the skeleton attaches to nearby control points. As the user poses the skeleton, the bones guide the associated control points, which in turn smoothly deform the geometry. In section IV we discuss an implementation of skeletal deformation. Using skeletal deformation, the designer can interact with an intuitive skeletal frame, rather than trying to orchestrate dozens of control points manually. The underlying lattice is essentially invisible to the user, but its presence as an intermediary between the skeleton and the surface guarantees smooth deformations.

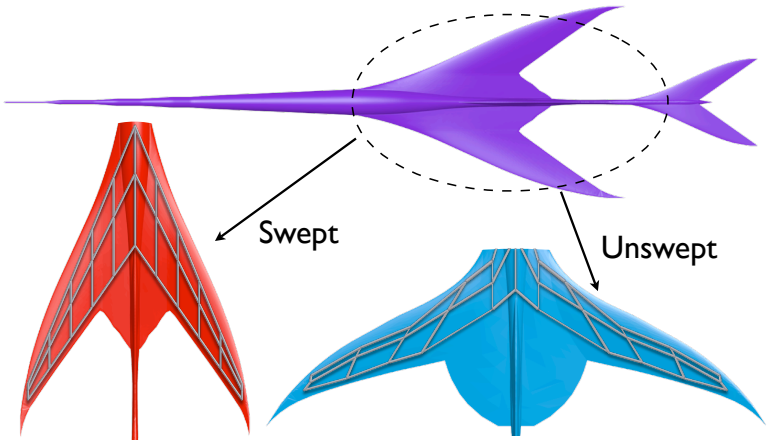


Figure 4: A skeleton adjusts the sweep of a supersonic wing. The underlying lattice-based deformation (not shown) is essentially invisible to the user.

2. *Constraint-Based Deformation*

An alternate tack is to allow the user to specify the displacements for a certain number of vertices on the geometry itself, called *pilot points*. The system then solves for a cage or lattice configuration that satisfies these positional constraints. This technique is termed *direct manipulation* because the user prescribes deformations directly on the surface, rather than indirectly through control points. Direct manipulation is more intuitive, as its parameters are located on a surface, rather than floating in space like in lattice and cage-based approaches.

By focusing only on a sparse collection of surface points instead of the entire surface, direct manipulation brings a helpful degree of surface-awareness to volumetric deformation, while avoiding the computational expense of standard surface-based methods. Moreover, there is great freedom to choose the number and location of design variables. Besides positional constraints, many other geometric constraints can be imposed, including volume, surface area, thickness, linearity and angular constraints. Yamazaki and Mouton developed a direct manipulation method for aerospace design.[26] Their implementation supports both FFD and radial basis functions as the underlying deformation method. In section V.B we present an implementation of constraint-based deformation for our platform.

IV. Deformation with Blender

Blender is a versatile open-source digital content creation suite. Its capabilities include geometry modeling, animation, shading, rendering and compositing.[1] Like most modern CG geometry tools[b], its native geometry format is discrete surfaces (e.g. triangulations or surface patches). Blender is designed for composing and rendering hundreds of thousands of visual scenes without user interaction in production environments. This focus on automation is also well-suited to aerospace design environments, where optimization frameworks require reliable geometry engines.

Custom plugins can be written with a unified API that provides Python handles to Blender's efficient geometry processing routines. A mature GUI facilitates interactive geometry setup. Both the scripting interface and the GUI invoke the same geometry kernel, so that any operation possible in the user interface is also scriptable. In the following sections, we distill Blender's dozen or so deformation methods down to the three most clearly useful for aerospace design.

A. Lattice Deformation

Lattice deformation is Blender's implementation of free-form deformation. A rectangular 3D lattice is placed around the geometry.[c] A volumetric B-spline deformation function is defined on the interior of the lattice, which controls the deformation of any geometry inside the volume. Figure 2 showed a lattice dramatically recontouring a fuselage. The number of control points in the lattice and its overall dimensions may be different for each coordinate direction, but the control points must be evenly spaced within each direction. If more precise control is needed over a particular region, a second localized lattice can be overlaid. This process is demonstrated in Figure 5, where one lattice exerts detailed control over an airfoil's leading edge, while a second lattice handles large-scale shape changes.

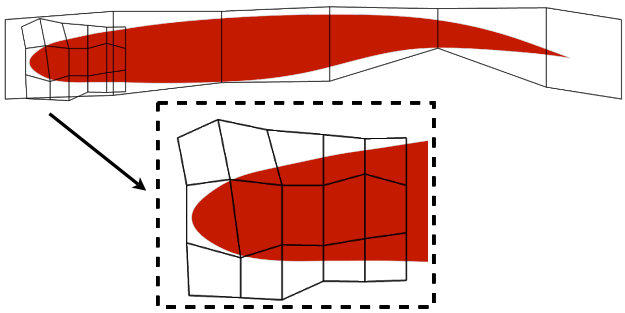


Figure 5: Two lattices shape an airfoil.

Lattice deformations have a localized effect on the geometry, because B-splines have finite support. As depicted in Figure 6, each control point has a region of influence of up to two control points away, and vertices outside this zone are unperturbed. Lattice deformation is rapid, even for complex geometries. Typical deformation timings on a modern laptop are given in Table 1.

[b] Appendix A compares several commercial alternatives to Blender, such as the widely-used Maya and 3ds Max suites.
[c] The lattice may also degenerate to 2D or even 1D to allow symmetric deformation.

B. Cage-based Deformation

Blender's cage-based deformation method is based on work at Pixar by Joshi et al.[27] From the user perspective, the process is visually similar to lattice deformation. The geometry is enclosed in a simple closed surface mesh. The surface is then embedded in the volume (discussed in section III.B.2). The embedding involves solving Laplace's equation on the interior of the cage and then using this solution to determine the influence of each control point on different regions of the geometry. Moving the control points deforms any geometry located within the cage. Figure 3 showed a cage reshaping the nose of a reentry capsule using Blender's implementation.

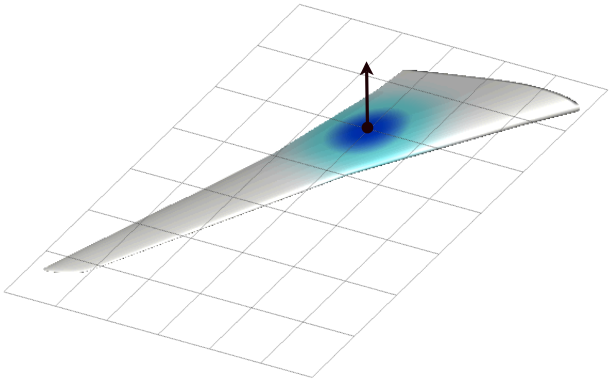


Figure 6: Surface sensitivity to a lattice control point. Dark blue corresponds to the highest sensitivity, light gray indicates zero sensitivity. This 2-D lattice could be used for planform deformation. To shape airfoil cross-sections, a full 3-D lattice would be used.

Each control point is used as a design variable. Unlike the regular lattice, the cage is free to take any shape, so long as it forms a closed surface. The designer can choose the number of design variables and is free to place the control points in any configuration, allowing detailed control over certain regions and coarser control over others. Because of the global nature of elliptic PDEs, each control point theoretically influences the entire geometry. In practice the sensitivity to a control point decays rapidly with distance, as was depicted in Figure 3. Once the geometry is embedded in the volume, deformations are extremely rapid, requiring only matrix-vector products. Typical timings are given in Table 1. Cages generally have fewer control points than lattices, because the cage covers a surface whereas a lattice fills a volume.

Table 1: Typical deformation times (in seconds) on a Core 2 Duo laptop. Timings are averaged over 10,000 deformations.

	27-point Lattice	**26-point Cage**	
Vertices	Deformation	Embedding*	Deformation
1.3 K	0.002	1.20	0.0017
6 K	0.009	1.24	0.0020
48 K	0.073	2.17	0.0024
127 K	0.191	6.20	0.0030
185 K	0.289	6.55	0.0029

* Cage embedding happens only once, during pre-computation. Subsequent deformations use the same stored embedding. The embedding time also depends on the resolution of the volume discretization.

C. Skeletal Deformation

In Blender's skeletal deformation method, a connected configuration of bones is embedded in the geometry. Though the bones can be linked directly to the discrete surface, this typically produces non-smooth deformations near joints. We instead define a two-phase deformation, where the bones deform a volumetric control grid (either a lattice or a cage), which in turn smoothly deforms the geometry.

This method was illustrated in in Figure 4, where a skeleton swept a supersonic wing forward. With skeletal deformation, intuitive CAD-like parameters can be used to orchestrate large-scale parametric deformations that can mimic or expand the original constructive parameterization. Taking advantage of Blender's ready extensibility, we created a custom plugin for interactive wing parameterization. The tool automatically generates multi-section wing skeletons based on standard design parameters including taper, span, chord, twist, sweep and dihedral.

V. Geometry Platform

Our geometry platform serves two functions. During problem setup, it can be used as an interactive geometry parameterization tool. Once the design parameters are chosen, it serves as an automated deformation engine, playing a role analogous to a CAD server. An optimization framework can request deformations through Blender's scripting interface. In addition to deformations, our platform also provides surface sensitivities for gradient-based design (examples of surface sensitivities were visualized in Figures 3 and 6). As Blender is open-source it would be possible to compute the surface sensitivities by analytically linearizing its deformation algorithms, but we have found that finite-differencing gives excellent and reliable results. Dataflow is managed with the XDDM protocol,[30] which simplifies integration with client design frameworks and provides access to a set of standard services including namespace management, symbolic function manipulation, and surface mesh differentiation.

Geometry Pipeline: Often a single geometry package is insufficient to allow a designer to fully express design intent. The designer may wish to exploit attractive features of multiple modelers by chaining them into a *geometry pipeline*, or sequence of geometry processing operations. Constructive modelers and deformation tools can be used in tandem to achieve the desired shape. For example, a geometry pipeline may begin with a CAD system that instantiates and triangulates a baseline nacelle. In turn, this triangulation may be deformed with a lattice or cage and then intersected with a wing and pylon generated by other modelers. Because geometry pipelines may be arbitrarily complex, we ensure that our platform can equitably sit either upstream or downstream of other geometry tools. Parameters from a constructive modeler can be used alongside deformation parameters from our platform, helping a designer achieve full design intent.

Extensibility: An important attribute of any general-purpose geometry platform is the ability to expand its functionality. While many deformation methods are native to Blender, this selection is not comprehensive. Every aerospace design group develops its own particular geometry requirements, which even the best geometry tools cannot be expected to satisfy. Extensibility is invaluable in a geometry platform, because it enables design groups to tailor a versatile and robust geometry tool to meet their needs by writing plugins instead of coding a geometry manipulation tool from scratch. Blender's API enables a designer to prototype virtually unlimited geometry deformations, but always within the sandbox of a reliable deformation engine. However, from the user perspective, the API is optional. It need only be used in cases where more particular control is required.

Supportability: Extensibility also ensures that a geometry tool will remain useful in the future. As new deformation techniques are developed both by aerospace groups and in the computer graphics industry, these methods can be integrated as plugins to a standard discrete geometry platform. We have developed several custom plugins, including a triangle subdivision tool,[8] file converters for various discrete geometry formats, and a tool that lets Blender serve as a GUI and visualizer for arbitrary third-party modelers. In the following sections we describe in detail two other major extensions to Blender developed as part of this work.

A. Interactive Parameterization

Taking advantage of Blender's readily extensible GUI, we have developed a set of interactive geometry parameterization tools. The tools allow deformation modes and design variables to be quickly prototyped in preparation for a design optimization study. In many design environments, the geometry parameterization phase can be a frustrating exercise in juggling cryptic text files, poorly documented deformation codes, format converters and visualization tools. Our platform encapsulates the entire process in one visual environment, helping the designer to quickly hone in on a good parameterization by showing precisely how a set of parameters will deform the geometry.

Specifically, we have developed a wing skeleton prototyping tool, which allows the user to specify sectional span, twist, sweep and dihedral, using a skeleton conforming to the wing planform. The wing skeleton displayed in Figure 4 was prototyped in seconds using this tool, as was the skeleton used for planform deformation in the examples section. We have also developed automatic cage and lattice generators, which created all of the cages and lattices shown in this paper, such as the cage in Figure 3.

B. Pilot Points: Constraint-Based Deformation

We present here a basic constraint-based deformation technique (introduced in section III.2), which we have implemented as a Blender plugin. In this method, the geometry is reshaped using a small set of "pilot points" located directly on the surface as design variables. From the designer's perspective, using pilot points is far simpler than trying to orchestrate clouds of control points floating in space.

By construction, any user-specified geometric constraints are intrinsically included in the parameterization. Put another way, we automatically restrict the design space to include only shapes satisfying the constraints. The parameterization inherently cannot produce an invalid shape (as defined by the user). Specifically, given a set of design variables $\mathbf{X}$ and constraints, we compute an intermediate deformation mapping $\mathbf{D}$ to obtain a new surface $\mathbf{T}$ that satisfies the constraints. Smooth deformations are ensured by using a lattice or cage as the intermediate deformer, which is automatically generated and essentially invisible to the user.

Each design variable $\mathbf{X}$ is treated by the solver as a constraint[d] f_i, which is a function of the surface $\mathbf{T}$ (e.g. a triangulation) and possibly also of the deformation parameters in $\mathbf{D}$ (e.g. control points of a cage or lattice):

$$f_i\left(\mathbf{T}\left(\mathbf{D}\right),\mathbf{D}\right)=\mathbf{X} \tag{1}$$

We do not directly control the entire surface, only a sparse collection of points on it. To solve for a deformation mapping $\mathbf{D}$ satisfying (or most nearly satisfying) the system of constraints f_i, we now write the system as a matrix equation using the notation $\mathbf{F}=[f_1,f_2\ldots f_n]^T$:

$$\mathbf{FD}=\mathbf{X} \tag{2}$$

and solve iteratively using

$$\Delta\mathbf{D}=-\mathbf{K}\left(\frac{\partial\mathbf{F}}{\partial\mathbf{D}}\right)^{-1}\Delta\mathbf{X} \tag{3}$$

where $\Delta\mathbf{X}$ is the difference between the current and target design variable values (i.e. the deviation from a satisfied set of constraints), and $\mathbf{K}$ is a diagonal matrix that controls the step size for each constraint independently. For a fully linear system of constraints with respect to the surface vertices, $\mathbf{K}$ can be set to the identity matrix, and the algorithm will reach the solution in one step. But if any constraint in the system is nonlinear, the solver must iterate, and $\mathbf{K}$ will affect convergence speed and stability.

The term $\frac{\partial\mathbf{F}}{\partial\mathbf{D}}$ is the sensitivity of the constraints to the deformation mapping. Note that the matrix $\frac{\partial\mathbf{F}}{\partial\mathbf{D}}$ is not square in general, so Equation 3 implies a pseudo-inverse. For under-constrained systems, this returns the perturbation vector $\Delta\mathbf{D}$ of smallest magnitude. For over-constrained systems, it returns the least-squares fit with the smallest overall constraint error. To compute $\frac{\partial\mathbf{F}}{\partial\mathbf{D}}$, we first expand it into a product of independent terms

$$\frac{\partial\mathbf{F}}{\partial\mathbf{D}}=\frac{\partial\mathbf{F}}{\partial\mathbf{T}}\frac{\partial\mathbf{T}}{\partial\mathbf{D}} \tag{4}$$

The term $\frac{\partial\mathbf{F}}{\partial\mathbf{T}}$ is the sensitivity of the constraint values to the geometry. It is derived analytically for each type of constraint. The second term, $\frac{\partial\mathbf{T}}{\partial\mathbf{D}}$, is the sensitivity of the surface to the deformation mapping.

While at first glance the matrix inversion in Equation 3 may appear expensive, in fact the size of the matrix is manageable. The two dimensions of $\frac{\partial\mathbf{F}}{\partial\mathbf{D}}$ are the number of design variables (i.e. pilot points) and the number of parameters in $\mathbf{D}$ (e.g. the number of lattice control points). Importantly, the dimensions of $\frac{\partial\mathbf{F}}{\partial\mathbf{D}}$ are independent of the surface complexity.

For gradient-based design, we also provide the sensitivity of the entire surface to the sparse set of pilot point design variables

$$\frac{\partial\mathbf{T}}{\partial\mathbf{X}}=\frac{\partial\mathbf{T}}{\partial\mathbf{D}}\left(\frac{\partial\mathbf{F}}{\partial\mathbf{D}}\right)^{-1} \tag{5}$$

[d]To the user and to the optimizer, the pilot points are viewed as design variables. But to the deformer, which is provided with a particular set of pilot point locations, these locations are considered constraints that must be satisfied.

Both terms in this product are computed during the constraint solve, so computing the surface sensitivities requires only an inner product of existing matrices.

We use this deformation technique to internally enforce geometric constraints frequently used in aerodynamic shape optimization. For example, thickness can be constrained by prescribing the location of one surface vertex relative to another vertex:

$$\mathbf{v}_i - \mathbf{v}_j = \mathbf{d}^* \tag{6}$$

Alternately, setting $\mathbf{v_j}$ to 0 allows the absolute location of vertex $\mathbf{v_i}$ to be prescribed, which establishes vertex $\mathbf{v_i}$ as a pilot point. The constraint sensitivities for use in Equation 4 are

$$\frac{\partial f}{\partial \mathbf{D}} = \frac{\partial \mathbf{v}_i}{\partial \mathbf{D}} - \frac{\partial \mathbf{v}_j}{\partial \mathbf{D}} \tag{7}$$

Other constraints, such as volume, surface area, projected area, or surface smoothness, are derived in a similar manner.

VI. Results

We test our geometry platform on several design examples, using a recently-developed aerodynamic design framework.[30] The design framework uses an embedded-boundary Cartesian mesh method for flow solutions. Objective gradients are computed using an adjoint formulation. Optimization is handled primarily with SNOPT[31] and in some cases with a BFGS quasi-Newton method[32] available in the design framework.

A. 2D Inverse Design with Lattice

The first test is intended to verify the consistency and reliability of our geometry platform on a problem with a known solution: inverse aerodynamic design for an airfoil in subsonic flow (Mach 0.57). We prescribe an attainable target pressure distribution and examine the convergence from an initial perturbed geometry to this target.

1. *Parameterization and Objective*

Figure 7 shows the parameterization. The airfoil is enclosed in a 10×2 lattice. At each of the 10 chord stations, the two control points are linked into orthogonal deformation modes corresponding to thickness and camber. Thickness is changed by moving the two control points in opposite directions. Camber is obtained by moving them in the same direction. Of the 20 thickness and camber parameters, nine serve as active design variables. The initial shape was obtained by randomly perturbing each active variable.

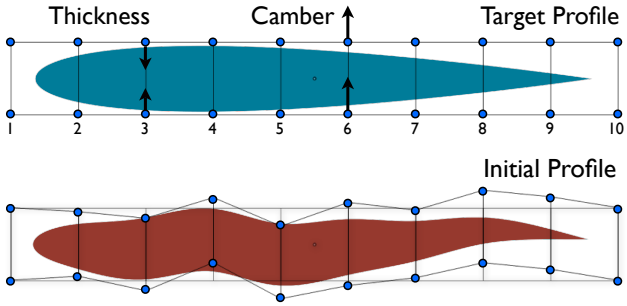


Figure 7: Lattice parameterization for inverse airfoil design problem. Thickness and camber are set at each of ten stations.

The target geometry is the NACA 0012 airfoil at 3.1° angle of attack. We specify a least-squares objective function to minimize the error between the actual pressure distribution and the target:

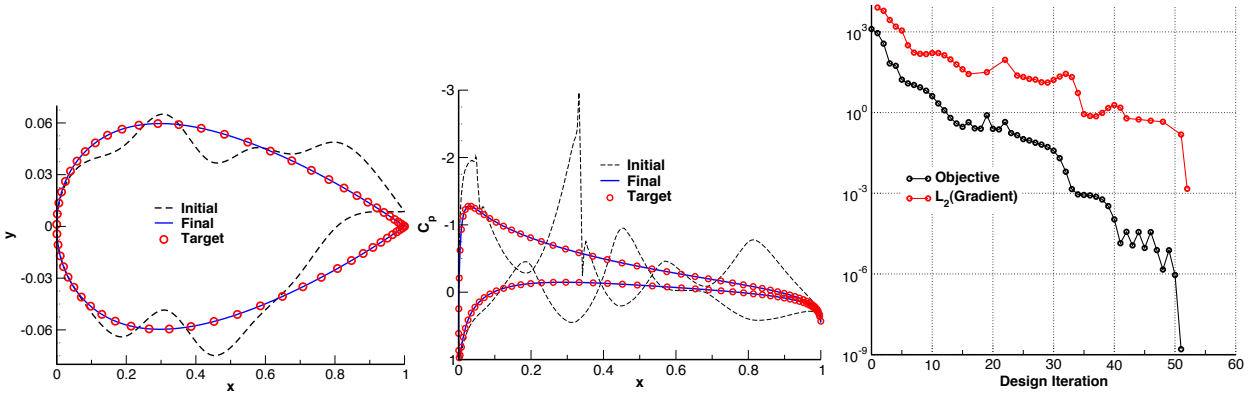
$$\mathcal{J} = \sum_{i=1}^{nverts} \left(C_p - C_p^*\right)_i^2 \tag{8}$$

where the target pressure coefficient C_p^* is specified at each vertex on the triangulation. As we used the same parameterization to generate the target and initial shapes, we expect that the optimization should recovery the target shape exactly.

2. *Optimization Results*

Figure 8 shows the successful recovery of the target airfoil and pressure distribution. As shown in Figure 8c, the objective function was reduced by 12 orders of magnitude over 50 design iterations, indicating recovery of the target pressure distribution to machine zero. The deep convergence verifies that our platform performs accurate and repeatable deformations of discrete geometries.

Flow and adjoint solutions were performed on meshes of about 12,000 cells. Each design iteration required at most 240 seconds on a current generation desktop CPU.[e] Computation of surface sensitivities required about 7 seconds per design variable, using the automatic surface mesh differentiation in the XDDM toolset. Similar studies using our platform's internal mesh differentiator cut the geometry processing time roughly in half.



(a) Airfoil (5× exaggerated thickness) (b) Pressure coefficient distribution (c) Objective convergence history

Figure 8: 2D inverse design results (Mach 0.57, $\alpha = 3.1°$).

B. Shape Matching

In this example we drive a NACA 0012 airfoil to a target RAE 2822 airfoil using two different deformation techniques. We solve the shape-matching problem first using our platform's lattice deformation technique (shown in Figure 9) and then using MASSOUD,[24] thereby benchmarking Blender's lattice deformer against a tool specifically developed for aerodynamic shape design.

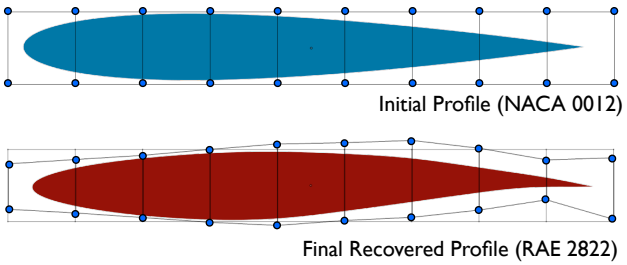


Figure 9: Blender lattice parameterization for shape-matching problem.

1. *Parameterization and Objective*

We first enclose the airfoil in a lattice, as shown in Figure 9. The lattice is identical to the one used in the first example, except that now all 20 thickness and camber parameters are used as active design variables. Next, we next create an analogous parameterization in MASSOUD, using a control net with 10 chord stations, aligned with the Blender lattice control points. Thickness and camber can be set at each station, for a total of 20 active variables. The purely geometric shape-matching objective is a sum of distances between corresponding vertices:

$$\mathcal{J} = \sum_{i=1}^{nverts} (\mathbf{v} - \mathbf{v}^*)_i^2 \tag{9}$$

where $\mathbf{v_i}$ are the current vertex coordinates and $\mathbf{v_i^*}$ are the corresponding target vertex coordinates.

[e] Xeon Westmere processor

2. *Optimization Results*

Both Blender and MASSOUD closely recovered the RAE airfoil. Figure 9 shows the final airfoil and lattice configuration reached by the Blender parameterization after about 50 design iterations. Figure 10 compares the best airfoils produced by the two methods. The frame on the right of Figure 10 shows a closeup of the trailing edge, where the largest difference between the final shapes is apparent. Some discrepancy is indeed expected, as Blender's lattice deformation and MASSOUD's NURBS surfaces are mathematically related but not identical.

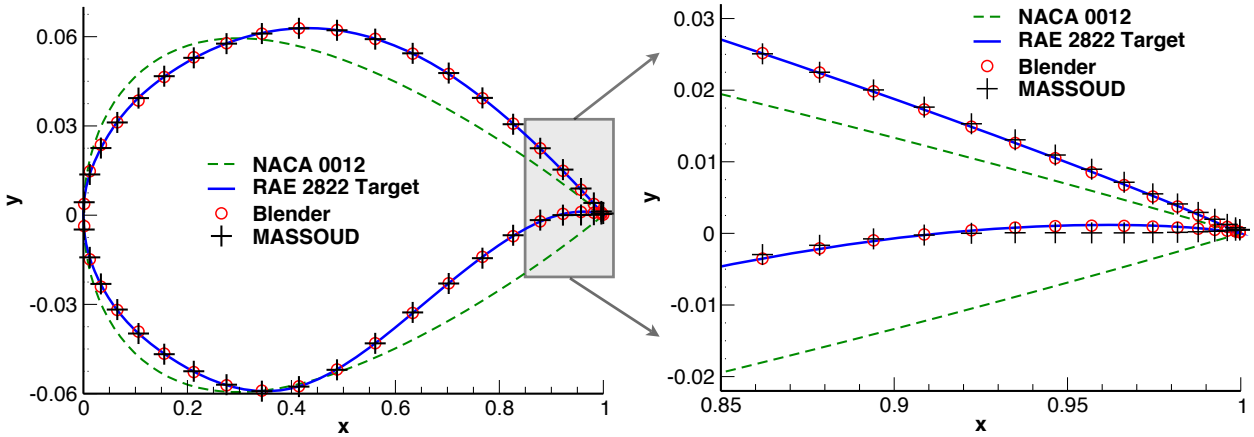


Figure 10: Shape-matching results. Left: Full airfoil (6× exaggerated thickness), Right: Trailing edge.

Figure 11 shows the convergence of the shape-matching objective function. The MASSOUD parameters converged more rapidly, but the Blender parameters achieved a convergence about two times deeper. The discrepancies at the leading and trailing edges shown on the right side of Figure 10 contribute to this difference in objective convergence. More control points would reduce the error further. But even with a modest number of design variables, the shape recovery with both techniques is quite good, showing that Blender can provide functionality similar to that of MASSOUD.

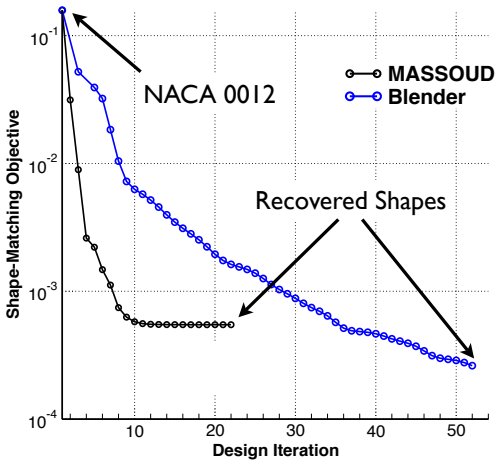


Figure 11: Objective convergence history for shape-matching example.

C. Constrained Airfoil Design with Pilot Points

In this example we optimize an airfoil for flight at two Mach numbers using the direct-manipulation deformation technique developed in section V.B. We impose thickness constraints and seek to minimize drag while maintaining lift.

1. *Parameterization and Objective*

Figure 12 shows the airfoil parameterization. Several "pilot points" on the upper and lower surfaces of the airfoil are used as design variables. Thickness constraints are added at 30% and 70% chord by linking the corresponding points on the upper and lower surfaces using Equation 6. These two stations are free to relocate horizontally and vertically, but the parameterization intrinsically preserves the thicknesses throughout the optimization. A background 12×5 lattice serves

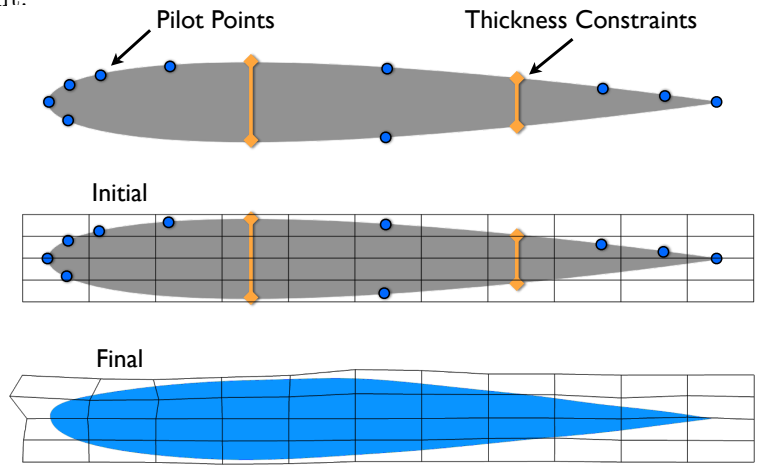


Figure 12: Pilot points parameterization.

as the intermediate deformer (called **D** in section V.B). Our constraint-based deformation plugin solves the system of constraints for a new lattice configuration, which generates a new surface satisfying the constraints. As the design variables and constraints are all linear, each deformation takes only a single iteration.

The pilot points at the leading and trailing edge are held fixed. The remaining pilot points can move vertically, and a few near the leading edge can also move horizontally. In total, there are 16 geometric design variables, in addition to the angle of attack at each design point. We seek to minimize drag while maintaining lift at two design points. The objective function is a weighted sum of lift and drag functionals:

$$\mathcal{J} = C_{D1} + 10\left(1 - \frac{C_{L1}}{C_L^*}\right)^2 + C_{D2} + 10\left(1 - \frac{C_{L2}}{C_L^*}\right)^2 \tag{10}$$

where C_L^* is the target lift coefficient. The two design points are Mach 0.7 and Mach 0.74. The target lift coefficient is set to 0.56 at both design points, which is equal to the initial airfoil's lift coefficient (based on a unit planform) at Mach 0.7 and 3° angle of attack.

2. *Optimization Results*

Figure 13 shows the optimized airfoil shape after 30 design iterations. The thickness constraints at 30% and 70% chord were necessarily met at every design iteration, because they were intrinsic to the parameterization. The 30% chord station shifted slightly downward, while the 70% chord station moved upward. As shown in Table 2, lift was maintained at both design points, and the optimizer (SNOPT) substantially reduced the drag at both Mach numbers. Figure 14 shows the weakened transonic shocks at both design points. The waviness in the optimized pressure distributions is due to our particular choice of pilot points. Additional design variables near the trailing edge on the lower surface could give the optimizer freedom to improve the aft-loading of the airfoil. Nevertheless, this case successfully demonstrates the handling of geometry constraints in constraint-based deformation.

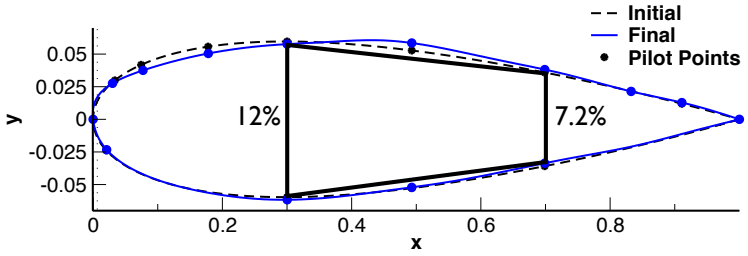
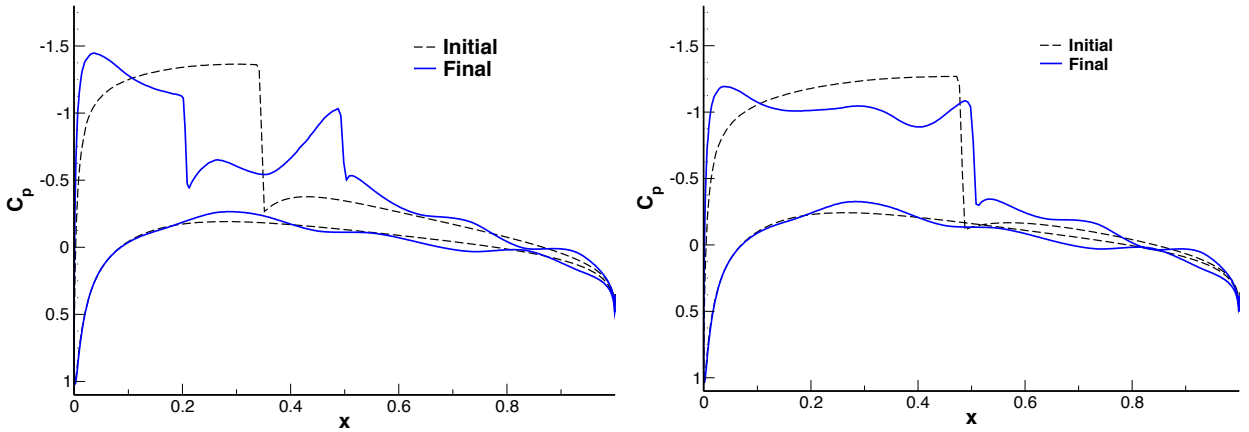


Figure 13: Optimized airfoil, showing pilot points and thickness constraints at 30% and 70% chord.

Table 2: Airfoil drag minimization results.

	Mach 0.70			Mach 0.74		
	C_L	C_D	L/D	C_L	C_D	L/D
Initial	0.56	0.0094	59	0.56	0.0202	28
Optimized	0.55	0.0033	167	0.56	0.0034	164



(a) M 0.7 (final $\alpha = 2.7°$)

(b) M 0.74 (final $\alpha = 2.3°$)

Figure 14: Pressure profiles for pilot points design problem.

D. Transonic Wing-Fuselage Integration with Skeleton and Lattice

In this example we perform drag-minimization at fixed lift for a transonic wing-body configuration at Mach 0.78. This example demonstrates the use of skeletal deformation for planform variations in conjunction with lattice deformation for airfoil deformation and fuselage contouring.

1. *Parameterization and Objective*

Figure 15 shows the initial geometry and the component-wise parameterization. The wing triangulation has about 86,000 vertices. Its shape is controlled by a lattice and a skeleton. The two deformations are applied sequentially. First, a $7 \times 4 \times 2$ lattice modifies the airfoil cross-sections. We constrain the control points in the lattice to move only in the normal direction, though in general they can move in any direction. B-splines loft the deformation between the four spanwise sections. For this example, we chose to demonstrate the maximum flexibility of the lattice deformer by using all but 10 of the 56 lattice control points as design variables. For applications that require fewer design variables, the control points can be linked as demonstrated earlier in the first two airfoil design examples. After the lattice deformation is applied, a skeleton reconfigures the planform by setting span, sweep, twist and dihedral at four stations.

The fuselage triangulation has about 56,000 vertices and is re-contoured with a $2 \times 2 \times 15$ lattice. Sectional radius parameters are created by linking the 2×2 square of lattice control points at each of the 15 axial stations along the fuselage. Finally, the wetted surface of the wing and fuselage configuration is extracted to form a CFD-ready triangulation.

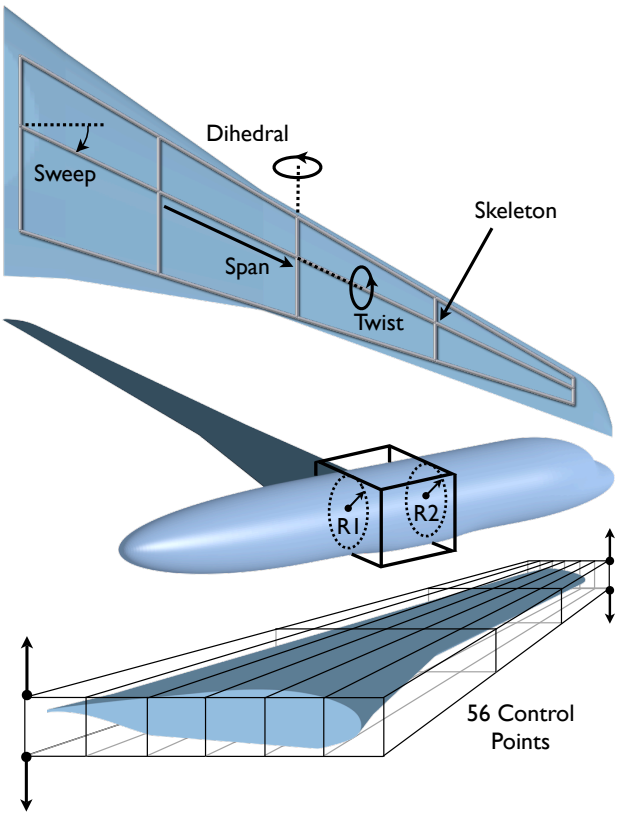


Figure 15: Wing-fuselage parameterization. Top: Skeleton controlling wing planform. Center: Lattice controlling fuselage radius (only two stations shown). Bottom: Lattice controlling airfoil shape.

To demonstrate the full flexibility of our geometry platform, we use nearly all the geometric parameters as active design variables. Span and dihedral are held fixed, and we allow only three fuselage radii near the wing root to change. For this transonic design problem, we expect the drag to be most sensitive by far to the wing lattice variables, which control the airfoil shape. Nevertheless, we include the fuselage and planform parameters to demonstrate the full range of deformations possible using our platform. With the angle of attack as the final variable, there are 58 active design variables in all. The objective function is

$$\mathcal{J} = C_D + 5\left(1 - \frac{C_L}{C_L^*}\right)^2 \tag{11}$$

with the target lift C_L^* set to the baseline shape's lift coefficient[f] at the initial angle of attack.

2. *Optimization Results*

Table 3 summarizes the aerodynamic improvement after 20 design iterations. Lift was maintained, while drag was reduced by 22 counts. Figure 16 shows the changes in pressure distribution across the wing planform and at three span stations. From the wing root to the trailing edge extension break (at 42% semispan), the flow has become essentially shock-free. While not as dramatic, shock reduction is also clearly evident

[f]Lift and drag are computed on the entire wing-fuselage configuration. The reference area is always the planform area of the original wing.

outboard along the wing's span. Because the chord dimension of the rectangular lattice was sized to fit the large wing root, the lattice was too sparse near the tip. The lattice resolution was too coarse to reshape the outboard region as effectively as the inboard sections.

The surface sensitivities to the design variables were computed in parallel on 24 processors. Blender is open-source, avoiding the software license limits that often prevent parallel scalability with commercial codes. Each design variable can be processed by its own instance of our geometry platform. This example shows that our platform can readily manage a combination of local and global shape changes. It also demonstrates our platform's facility with exercising precise control over local shape changes, a necessary capability for transonic wing design.

Table 3: Wing-body integration results.

	C_L	C_D	L/D
Initial	0.313	0.0148	21.2
Optimized	0.313	0.0126	24.7

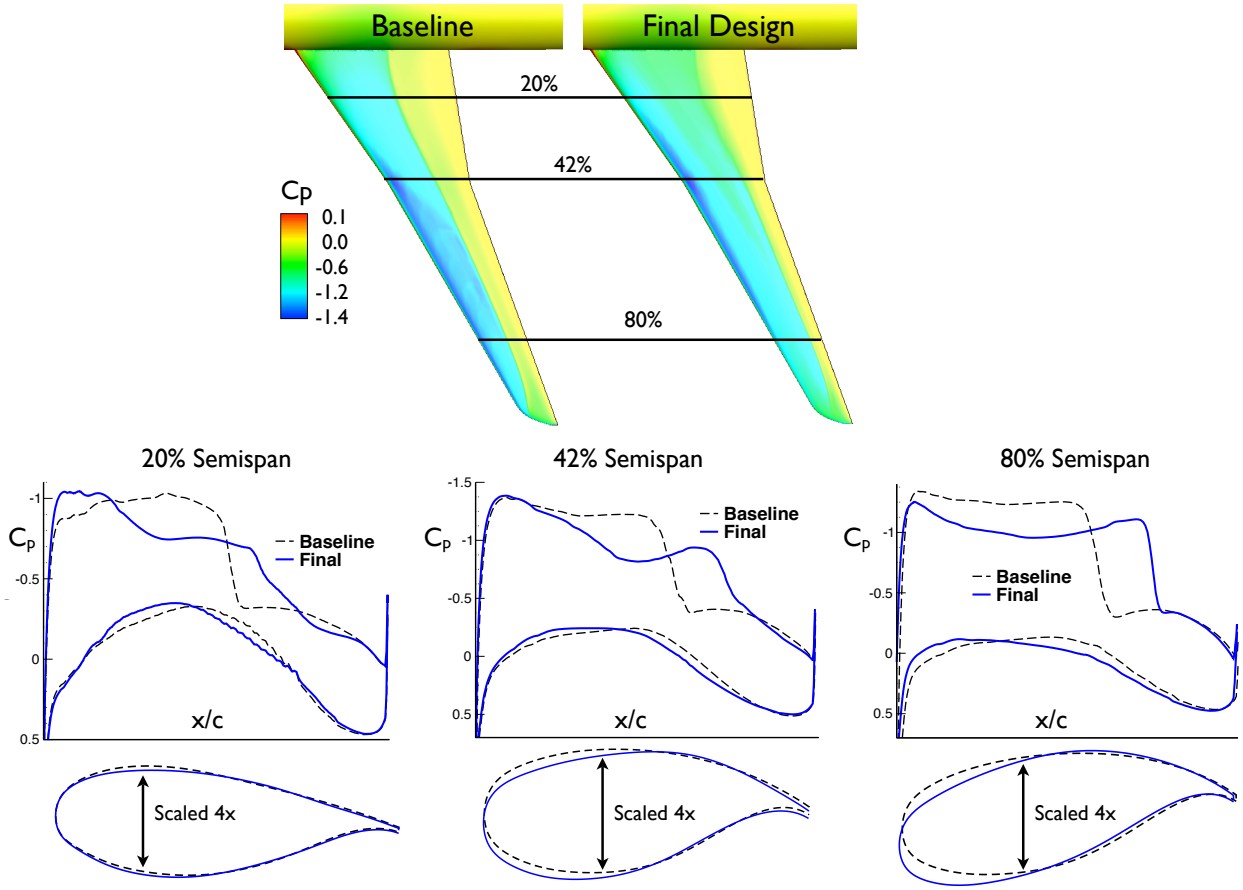


Figure 16: Aerodynamic results for wing-body integration problem.

E. Geometry Pipeline Example

In this example we use our platform to augment the functionality of an existing constructive modeler. We start with a wing generated by RAGE[g], a commercially available constructive aerospace geometry tool. The constructive model controls the wing shape with hundreds of design parameters. While these parameters allow instantiation of a wide range of shapes, they do not directly permit certain deformation modes of interest. Instead of attempting to re-parameterize the constructive model (which may or may not be possible, depending on the desired deformation), we add the necessary parameters using our platform as a second stage in a geometry pipeline.

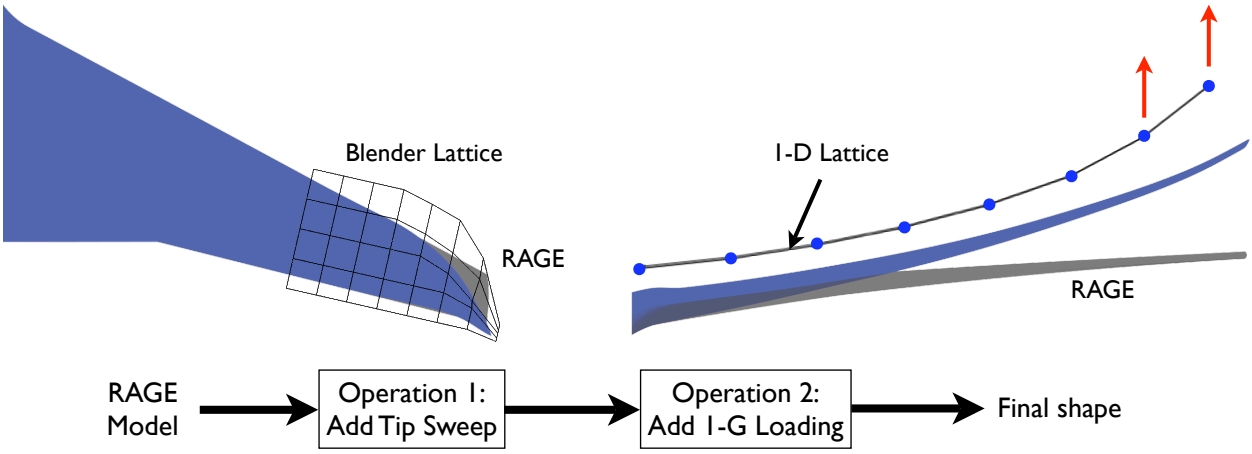


Figure 17: Geometry pipeline.

We consider two new deformation modes that are commonly either missing from constructive modelers or tedious to implement. The first gives the wingtip a parabolic sweep. We lay a 2D lattice over the wingtip to perform the necessary planform deformation, as shown in Figure 17. The second deformation mode is an approximation of 1-G flight loading dihedral, which we generate using a 1D lattice extending in the span direction.

We are able to use our platform as a post-processing tool for detailed modifications, while keeping the advantages of a powerful constructive modeling tool. The designer can express design intent more fully and simply by taking advantage of the strengths of each tool, rather than trying to force one tool to do everything. Design parameters from both the constructive modeler and our platform can be used simultaneously for shape optimization.

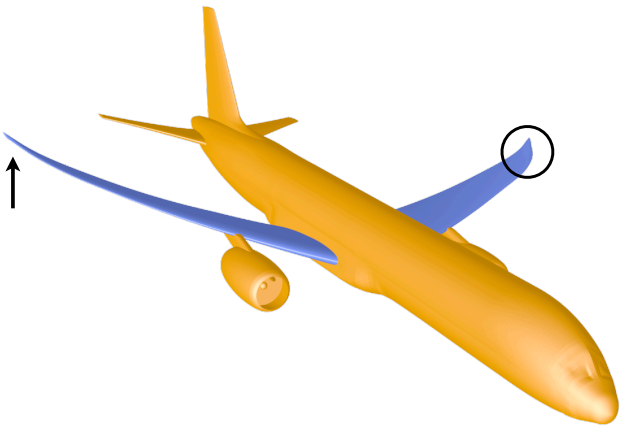


Figure 18: Final combined deformation.

VII. Summary and Conclusions

We have developed a platform for deformation of discrete geometry. The platform is implemented in the open-source tool Blender, capitalizing on tremendous investment by the CG industry in geometry techniques. The platform supports lattice and cage-based deformation, as well as intuitive methods such as skeletal deformation. Taking advantage of Blender's extensibility, we have implemented a direct manipulation deformation algorithm as a plugin. We have also developed interactive tools for rapidly prototyping aerospace geometry parameterizations. Geometry deformation is scriptable and fully automated, and the

[g]Rapid Aerospace Geometry Engine[33]

platform provides surface sensitivities along with the deformations. We tested our platform as an automated geometry engine on several aerodynamic shape optimization examples, evaluating its performance on 2D airfoil design and shape-matching problems and on a 3D wing-body integration problem. We showed that the platform can be pipelined with other geometry modelers to permit new modes of deformation.

Looking to the future, automated shape optimization is certain to play an ever-increasing role in design. From the computational standpoint, automated aerospace design environments bear close resemblance to large-scale rendering environments in the computer animation industry. Both depend on reliable, automated, and scriptable geometry tools. A cross-pollination of ideas would lead to far superior discrete aerospace geometry tools than exist today. In developing this platform, we hope to provide the aerospace community with basic geometry manipulation functionality that has long been considered standard by the computer graphics industry. We plan to develop a more comprehensive system for constraint-based direct manipulation and to further develop interactive parameterization tools. We also hope to test our tool on multidisciplinary deformation applications, such as aero-elasticity modeling.

Acknowledgments

The authors wish to thank NASA Langley for permission to use MASSOUD, and Desktop Aeronautics for permission to use RAGE. We are also are grateful to Antony Jameson (Stanford) for helpful discussions. This work was supported by the NASA Ames Research Center contract NNX09SE60G.

References

[1]www.blender.org.

[2]Hicken, J. E. and Zingg, D. W., "Aerodynamic Optimization Algorithm with Integrated Geometry Parameterization and Mesh Movement," *AIAA Journal*, Vol. 48, No. 2, February 2010.

[3]Désidéri, J.-A., Majd, B. A. E., and Janka, A., "Nested and Self-Adaptive Bezier Parameterizations for Shape Optimization," *Journal of Computational Physics*, Vol. 224, No. 1, May 2007, pp. 117–131.

[4]Duvigneau, R., "Adaptive Parameterization using Free-Form Deformation for Aerodynamic Shape Optimization," Tech. Rep. 5949, INRIA, 2006.

[5]Fudge, D. M., Zingg, D. W., and Haimes, R., "A CAD-Free and a CAD-Based Geometry Control System for Aerodynamic Shape Optimization," *43rd AIAA Aerospace Sciences Meeting and Exhibit*, No. 2005-0451, Reno, NV, January 2005.

[6]Kulfan, B. M., "Universal Parametric Geometry Representation Method," *J. Aircraft*, Vol. 45, No. 1, January 2008, pp. 142–158.

[7]Nemec, M. and Aftosmis, M. J., "Aerodynamic Shape Optimization Using a Cartesian Adjoint Method and CAD Geometry," *24th Applied Aerodynamics Conference*, No. 2006-3456, San Francisco, CA, June 2006.

[8]Persson, P.-O., Aftosmis, M. J., and Haimes, R., "On the Use of Loop Subdivision Surfaces for Surrogate Geometry," *15th Annual Meshing Roundtable*, October 2007.

[9]Dubé, J.-F., Guibault, F., Vallet, M.-G., and Trépanier, J.-Y., "Turbine Blade Reconstruction and Optimization Using Subdivision Surfaces," *44th AIAA Aerospace Sciences Meeting and Exhibit*, No. 2006-1327, Reno, NV, January 2006.

[10]www.integrityware.com/subdnurbs.html.

[11]www.gosculptor.com.

[12]Samareh, J. A., "A Survey of Shape Parameterization Techniques," *CEAS/AIAA/ICASE/NASA Langley International Forum on Aeroelasticity and Structural Dynamics*, Williamsburg, VA, June 1999.

[13]Hicks, R. M. and Henne, P. A., "Wing Design by Numerical Optimization," *J. Aircraft*, Vol. 15, No. 7, July 1978.

[14]Cliff, S. E., Thomas, S. D., and Hawke, V. M., "Swing-Wing Inline-Fuselage Transport Design Studies at Supersonic Flight Conditions," Vol. AIAA Paper 2009-7073, Hilton Head, SC, September 2009.

[15]Berkenstock, D. C. and Aftosmis, M. J., "Structure-Preserving Parametric Deformation of Legacy Geometry," *12th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, No. 2008-6026, Victoria, BC, September 2008.

[16]Botsch, M., Pauly, M., Gross, M., and Kobbelt, L., "PriMo: Coupled Prisms for Intuitive Surface Modeling," *Eurographics Symposium on Geometry Processing*, 2006.

[17]Botsch, M. and Sorkine, O., "On Linear Variational Surface Deformation Methods," *IEEE Transactions on Visualization and Computer Graphics*, Vol. 14, No. 1, January 2008.

[18]Gain, J. and Bechmann, D., "A Survey of Spatial Deformation from a User-Centered Perspective," *ACM Transactions on Graphics*, Vol. 27, No. 4, October 2008.

[19]Jakobsson, S. and Amoignon, O., "Mesh deformation using radial basis functions for gradient-based aerodynamic shape optimization," *Computers and Fluids*, Vol. 36, No. 6, July 2007, pp. 1119–1136.

[20]Morris, A. M., Allen, C. B., and Rendall, T. C. S., "Domain-Element Method for Aerodynamic Shape Optimization Applied to a Modern Transport Wing," *AIAA Journal*, Vol. 47, No. 7, 2009.

[21]Rendall, T. C. S. and Allen, C. B., "Unified Fluid-Structure Interpolation and Mesh Motion using Radial Basis Functions," *Int. J. Numer. Meth. Eng.*, Vol. 74, 2008, pp. 1519–1559.

[22]Allen, C. B. and Rendall, T. C. S., "CFD-Based Shape Optimization of Hovering Rotors Using Global and Local Parameters," *28th AIAA Applied Aerodynamics Conference*, No. 2010-4236, Chicago, IL, June 2010.

[23]Sederberg, T. W. and Parry, S. R., "Free-Form Deformation of Solid Geometric Models," *ACM SIGGRAPH*, Vol. 20, No. 4, August 1986.

[24]Samareh, J. A., "Multidisciplinary Aerodynamic-Structural Shape Optimization using Deformation (MASSOUD)," *8th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, No. 2000-4911, Long Beach, CA, September 2000.

[25]Samareh, J. A., "Aerodynamic Shape Optimization Based on Free-Form Deformation," *10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, No. 2004-4630, Albany, NY, July 2004.

[26]Yamazaki, W., Mouton, S., and Carrier, G., "Geometry Parameterization and Computational Mesh Deformation by Physics-Based Direct Manipulation Approaches," *AIAA Journal*, Vol. 48, No. 8, August 2010, pp. 1817–1832.

[27]Joshi, P., Meyer, M., and DeRose, T., "Harmonic Coordinates for Character Articulation," *Pixar Technical Memo*, 2007.

[28]Ben-Chen, M., Weber, O., and Gotsman, C., "Variational Harmonic Maps for Space Deformation," *ACM*, Vol. 28, No. 3, August 2009.

[29]Anderson, W. K., Karman, S. L., and Burdyshaw, C., "Geometry Parameterization Using Control Grids," *12th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, No. 2008-6028, Victoria, BC, September 2008.

[30]Nemec, M. and Aftosmis, M. J., "Parallel Adjoint Framework for Aerodynamic Shape Optimization of Component-Based Geometry," Vol. AIAA Paper 2011-1249, Orlando, FL, January 2011.

[31]Gill, P. E., Murray, W., and Saunders, M. A., "SNOPT: An SQP Algorithm for Large-Scale Constrained Optimization," *SIAM Journal on Optimization*, Vol. 12, 1997, pp. 979–1006.

[32]J E Dennis, J. and Schnabel, R. B., *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice Hall, Englewood Cliffs, NJ, 1983.

[33]Rodriguez, D. L. and Sturdza, P., "A Rapid Geometry Engine for Preliminary Aircraft Design," Vol. AIAA Paper 2006-929, Reno, NV, January 2006.

Appendix

A. Comparison of Geometry Tools

Blender's open-source status is highly advantageous in terms of cost and therefore scalability. Unlimited instances of the program can be active as geometry engines running in parallel, without facing the license limits typical of commercial software. Nevertheless, the computer graphics industry is quite large, and many powerful commercial tools are also available.

Table 4 compares key features of the most popular software packages. Scripting is available in every package, but only the more powerful suites are designed to be driven remotely in an automated design environment. Most of these more advanced packages also support Unix, a crucial feature for many aerospace design environments. While most of the geometry tools support more than one scripting language, Python is currently the most common. Each package has its own flavor and offers a different variety of deformation techniques and mesh processing routines. As a general rule, the more expensive packages are more comprehensive in this aspect. On the other hand, despite being free, Blender has a quite useful range of deformers and is competitive with the most powerful suites.

Table 4: Comparison of nine widely-used computer animation packages.

	Price*	Automated**	Unix	Python	Other languages
Blender	0	✓	✓	✓	-
Maya	3495	✓	✓	✓	MEL
3ds Max	3495	✓	✗	✗	MaxScript
SoftImage XSI	2995	✓	✓	✓	Perl, VB, JavaScript
Houdini	1995	✓	✓	✓	HScript, VEX
Lightwave	1495	✗	✗	✓	LScript
Cinema 4D	995	✗	✗	✓	COFFEE
Modo	995	✗	✗	✗	Perl, LUA
ZBrush	699	✗	✗	✓	ZScript

* Prices are in USD for a standard single-user license as listed on each product's website on 5 December 2011.

** Automated refers to the ability of the software to be called remotely in scripted batch mode.