Deformation of skeleton based implicit objects

Sondre Langeland Hisdal[*]
Supervised by: Julius Parulek[†]

Institute of Informatics
University of Bergen
Bergen / Norway

Abstract

In this paper we present a precise contact modeling environment for skeleton based implicit objects. To render the scene composed of these implicit objects, we have implemented the state-of-the-art raycasting algorithm, called marching points, on GPU using CUDA. Further, we introduce how to interactively deform the implicit objects when they collide. To achieve this we studied several ways to deform the objects. We implemented two well-known approaches, where we also proposed a new method created as combination of both approaches. Both these approaches as well as our method are described in this paper.

The implicit objects are implemented as distance surfaces. The field function for these only depend on the distance from the skeleton, and are easy to evaluate. We have achieved support for deformations of objects based on point and line skeletons.

Keywords: Implicit objects, Deformations, GPU, CUDA, Raycasting

1 Introduction

The goal of this project was to implement a precise contact modeling environment for skeleton based implicit objects. This work is an extention of a previous work with convolution surfaces, where convolution of skeleton primitives and an implicit kernel function was used to model merging of implicit objects. Figure 1 shows two spherical objects, made from point-skeletons, merging together. As shown in the figure you can see the objects stretch and merge together when they are brought closer to eachother.

In this project we were not interested in merging of objects, but the deformation of the objects when they collide. Implicit objects can be used to model organic structures and it is interesting to see how these behave when they intersect. It is neccessary to have a fast GPU based deformation of implicit objects if this is going to be interactive. By representing the objects with skeletons we save a lot of memory compared to using meshes. We have implemented support for point and line skeletons for this project. We looked at several ways to achieve deformation. We have implemented two well-known techniques, as well as introduced a new technique which is a combination of both techniques.

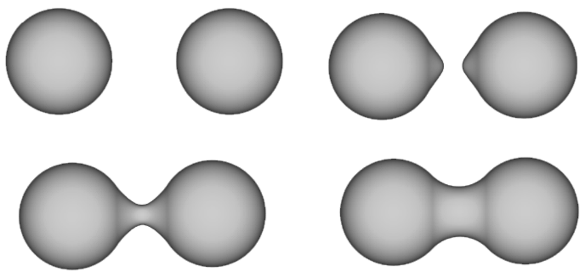


Figure 1: Figure shows two surfaces of point-skeletons merging together.

To achieve an interactive rendering we have implemented a raycasting algorithm called marching points on GPU. Calculation of deformations is very expensive and it needs to be parallelized to be interactive. The marching points algorithm is easily parallelized and well suited for rendering of implicit objects.

The rest of the paper is structured as follows. In Section 2 we discuss related work. In section 3 we describe the modelling and deformation in detail. In section 4 we describe the marching points algorithm. In section 5 we describe implementation and show results. Finally we conclude and talk about future work.

2 Related Work

2.1 Skeleton based implicit objects

Skeleton based implicit objects are objects defined by a skeleton primitive, like a point or a line, and some implicit function. The implicit function creates a object around the skeleton. We are on the surface of the object when the field function equals some iso-value, $f(p) - iso = 0$, where p is a position in space. By evaluating the field function from any position we can tell if the position is inside or outside the object. There are two main approaches to construct

[*]shi015@student.uib.no
[†]parulek@gmail.com

an implicit surface from a skeleton, distance surfaces and convolution surfaces. Distance surfaces use the distance from a point to the closest point on the skeleton when calculating the field function. Convolution surfaces integrate the contribution from all points on the skeleton when evaluating the field function. We have used distance surfaces for this project.

2.2 Deformation

For theoretical background for the deformations we have looked at several papers [1, 2, 3, 4, 5] about deformable objects. We have in particular looked at two of these papers[4, 2] for this project, and have implemented the techniques described in these. To our knowledge this has not been done on GPU before, and very little work has been done in this area the last years.

3 Deformations

Deformations should occur when two objects collide. To achieve this a deformation term is added to the field function around the intersection. We will have a surface when

$$f_i + deform(f_j) = iso \tag{1}$$

When looking at object i we calculate the field function of that object and add the deformation term caused by object j on object i. As a part of this project two different approaches for modeling deformations were implemented. We will describe each of these later in this section, but first we will look at how collision detection is done, since the same approach is used for both deformation techniques.

3.1 Collision detection

Before we can add a deformation term to the field function we need to check if the two objects actually intersect. We have done this by checking if the closest middle point between the two skeletons is inside both of the objects. This is easy to do for point skeletons. For point skeletons we just find the middle point between the two skeletons and check if that is inside both. For line skeletons it is a bit more difficult. The way it have implemented, we first find the shortest line between the two line skeletons, and then check if the middle of the line is inside both object. In figure 2 the shortest line segment between the two line segments AB and CD, is the line segment CE. If the objects defined by line segment AB and CD are intersecting, the middle point of CE, point F in the figure, will be inside both objects. For intersection between point and line skeletons we find the closest point on the line to the point and check the middle point of the line going from that point of the line to the point skeleton. This only works when the objects have the same width. If the skeletons have different widths we can not check the middle point, but the same basic approach can be used. First we find the shortest line between the skeletons and then instead of finding the middle we find the point that corresponds to the widths of the objects.

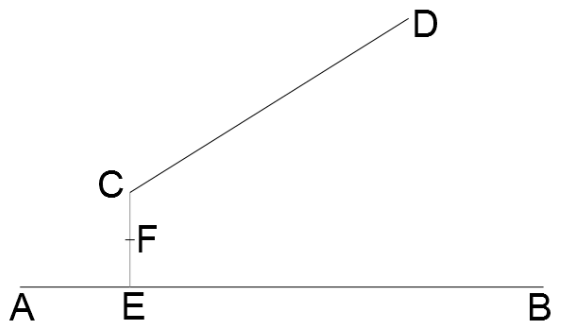


Figure 2: Showing closest middle point between two line segments.

3.2 Dual layer implicit objects

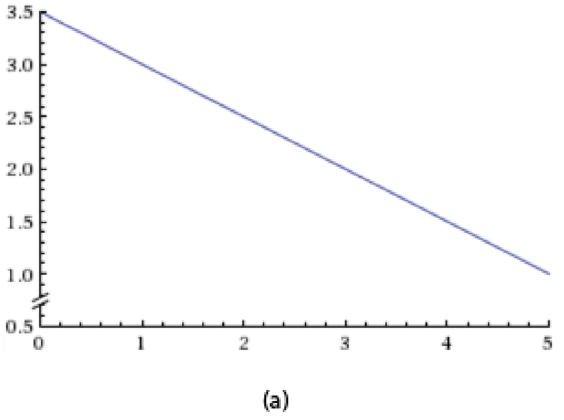
3.2.1 Object representation

The first technique that was implemented uses objects with two layers, one rigid inner layer and one deformable outer layer. The deformable layer starts where the rigid layer ends. This gives us two field functions for each object. The field functions depends on the distance, r, to the closest point on the skeleton. Figure 3 shows the profiles of the field functions for the two layers. To control the form of the objects we have three parameters. R is the scope of influence, meaning the longest distance away from the skeleton that should influence the object. r_0 is the thickness if the rigid layer, and k is the stiffness of the rigid layer. The surface of the object will be at the edge between the rigid and deformable layer.

$$f(r) = -kr + kr_0 + 1, \quad 0 \le r \le r_0 \tag{2}$$

$$\begin{aligned} f(r) = (r-R)^2 (&\frac{r(-k(r_0-R)-2}{(r_0-R)^3} \\ &+ \frac{kr_0(r_0-R)-R+3r_0}{(r_0-R)^3)}), \quad r_0 < r \le R \end{aligned} \tag{3}$$

Both these functions evaluate to 1 when $r = r_0$, giving a smooth transition from the rigid to deformable layer. r is the distance to the skeleton. This technique originally also computed forces between the intersecting objects. Computaion of forces has not been implemented, since the focus of this project was on the visual aspect of deformations alone. This technique is described by Gascuel in [4].



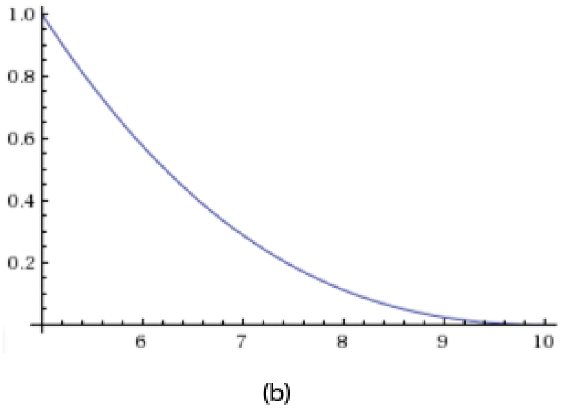


Figure 3: Profile of field function for rigid layer of the object (a) and for the deformable layer of the object (b).

3.2.2 Deformation term

As stated earlier, when two objects intersect they will be deformed by adding a deformation term to the field function around the intersection area. In this approach this deformation term is a function dependent on the distance to the object that is causing the deformation. See figure 4. Along with the distance it requires the maximum compression caused by the intersection. In areas that are inside both objects we need to compress the objects so they end up just touching and not intersecting each other. The maximum compression is used to see how large the deformations should be. Little compression gives little deformation. Besides this the function takes two parameters that allow you to control the deformation. Parameter w is the maximum distance that will be used and parameter *a* controls the height of the deformations along with the maximum compression term.

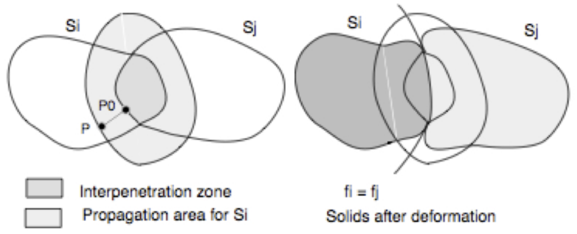


Figure 4: Distance used in deformation function. Figure by Gascuel [4].

$$
\begin{aligned}
deform &= 4\frac{wk-4a_0}{w^3}r^3 \\
&\quad +4\frac{3a_0-wk}{w^2}r^2+kr,\ \ 0<r\leq\frac{w}{2} \qquad (4)\\
deform &= \frac{4a_0(r-w)^2(4r-w)}{w^3},\ \ \frac{w}{2}<r\leq w
\end{aligned}
$$

$a_0 = a * maximal - compression$, r is the distance to the intersecting objects surface, and k is the stiffness value as described earlier. Figure 5 shows the profile of the deformation function. $a0$ is maximum of the function and is found when r equals $w/2$. There was a typo in the original paper that lead to some confusion. The deformation function from $w/2$ to w had a plus sign where it should be a multiplication. This lead to the two parts of the deformation function not matching at r equals $w/2$.

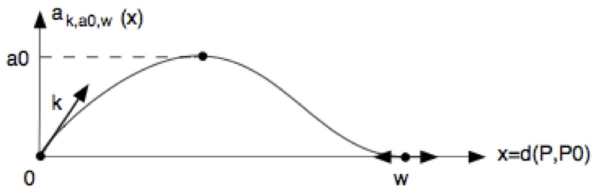


Figure 5: Profile of the deformation function. Figure by Gascuel [4].

3.3 Single layer implicit objects

3.3.1 Object representation

The second technique that was implemented uses a more simple way to represent the objects. It just uses one layer which field function is only dependent on the distance to the skeleton. The field function I have used is

$$f(r) = \frac{1}{r} - 1,\ \ 0 < r \leq 1 \qquad (5)$$

To support different widths of objects a width value k can be added and r in the field function be replaced with r/k. Figure 6 shows the profile of the field function.

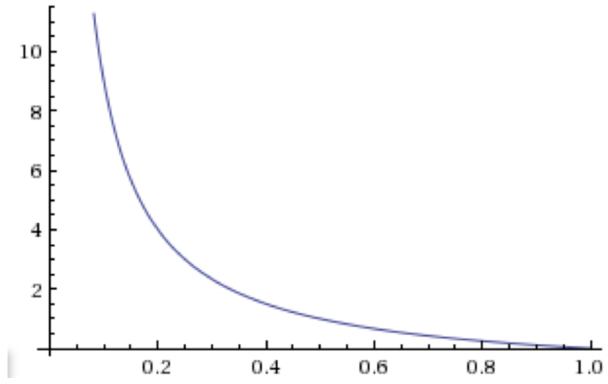


Figure 6: Profile of the field function used in the second technique.

3.3.2 Deformation term

This deformation technique was described by Angelidis et al. in [2]. This technique uses the field function of the intersecting object to deform the object. If object i and object j intersect, deformation of object i uses the field function, f_j , of object j. To control the deformation we have three parameters c_j, m, and h. c_j is the minimum value of f_j that should be part of the deformation area. m is the value of f_j where the top of the deformation should be. h is the maximum of the deformation function.

$$
\begin{aligned}
deform_i(f_j) &= (3-2\frac{f_j-c_j}{m-c_j})^2 h, \quad c_j \le f_j \le m \\
deform_i(f_j) &= h+((1-3h)+ \\
&2h-1)\frac{f_j-m}{iso-m})(\frac{f_j-m}{iso-m})^2, \quad m < f_j \le iso
\end{aligned} \tag{6}
$$

The profile of the deformation function can be seen in figure 7. In the paper by Angelidis et al. they also experimented with other deformation functions. This function is the one I found to be the best. It provides good control of the deformation without too many parameters. It should be noted however that there are other deformation functions that may be better depending on what is needed. For example, Angelidis et al. provided a function for creating ripples, in their paper[2].

3.4 Improved deformation of single layer implicit objects

A problem we found with the single layer technique was that it created too much deformation when the objects barely intersect. There was no smooth transition from no deformation to big deformation. The dual layer technique used the maximum compression term to control how big

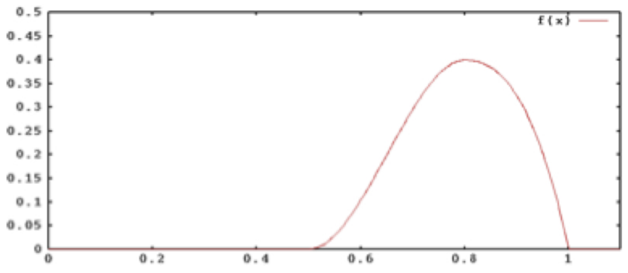


Figure 7: Profile of the deformation function used in the single layered technique. [2]

the deformation should be. By adding this to the single layer technique we were able to get a smoother transition. Instead of having a fixed maximum height of the deformation function we scale it by how much the object intersect.

4 Marching Points

Rendereing of implicit objects can be done by raycasting. Singh and Narayanan have introduced a ray casting algorithm for rendering implicit surfaces on GPU[6]. For each ray you sample the ray at a set interval looking for a root. Another way of looking at this is to have a point march down the ray and take samples, hence the name marching points. To find a root you do a sign test. You compare the current sample to the previous one and check if the sign has changed. If the sign has changed there is a root somewhere in the interval between the previous and the current sample. In figure 8 the sign will change from the sample at point B to the sample at point C. This means there is a root in the interval [B,C]. When an interval with a root is found the exact position of the root can be found using bisection. If a too large step size is used objects can be missed completely. If the object falls inbetween two sample points, the algorithm has no way of discovering the object. This makes the quality and effectiveness of the algorithm very dependent on the step size.

The rays cast using the Marching Points algorithm are independent of eachother. This allows us to cast multiple rays in parallel using the GPU. Using CUDA one thread processes one ray. For each ray the algorithm marches forward a small step at the time until it finds the interval where there is a root. When this interval is found, the root is found using bisection.

5 Implementation and Results

5.1 Framework

Our framework have been implemented using Python and CUDA. The raycasting and all evaluation of surfaces and deformations are performed on GPU using CUDA.

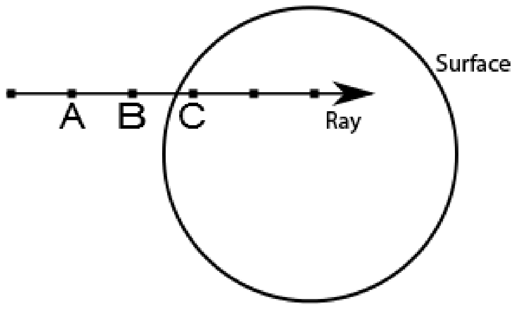


Figure 8: Illustration of Marching Points algorithm. A ray is shot towards the surface. Sampling occurs on regular intervals on the ray, like point A, B and C.

5.2 Dual layer implicit objects

The dual layer technique have been implemented as described without any alterations. To define the object we have used the parameters $k = 0.5$, $r_0 = 5$, and $R = 10$. To deform the object different parameters were tried but in the end $w = 10$ and $alpha = 0.3$ were used as default values. This technique gives very good control over the deformation and we can get deformations over a very large area if we want to.

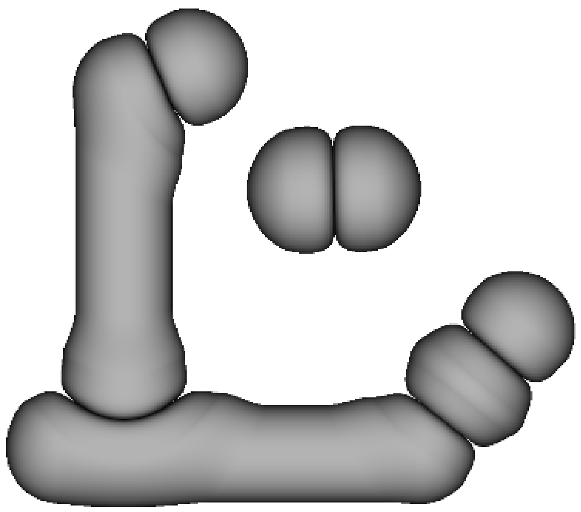


Figure 9: Rendering when using technique 1. Default parameters.

5.3 Single layer implicit objects

A problem with this technique was that the deformation when two objects barely intersect was too big. To fix this the height, parameter h, of the deformation was scaled by the maximum compression in the same way as in the dual layer technique. This gave a smoother transition from a scenario where no deformation should occur to a scenario where it should occur. This can be seen in figure 10. Figure 10 (a) does not have any scaling. This results in a too large deformation. In the figure the objects are barely intersecting and the deformation is already quite large. In figure 10 (b) scaling is applied. We downscale the height of the deformation when there is little intersection. This gives a much smaller deformation which is more fitting to how much the objects intersect.

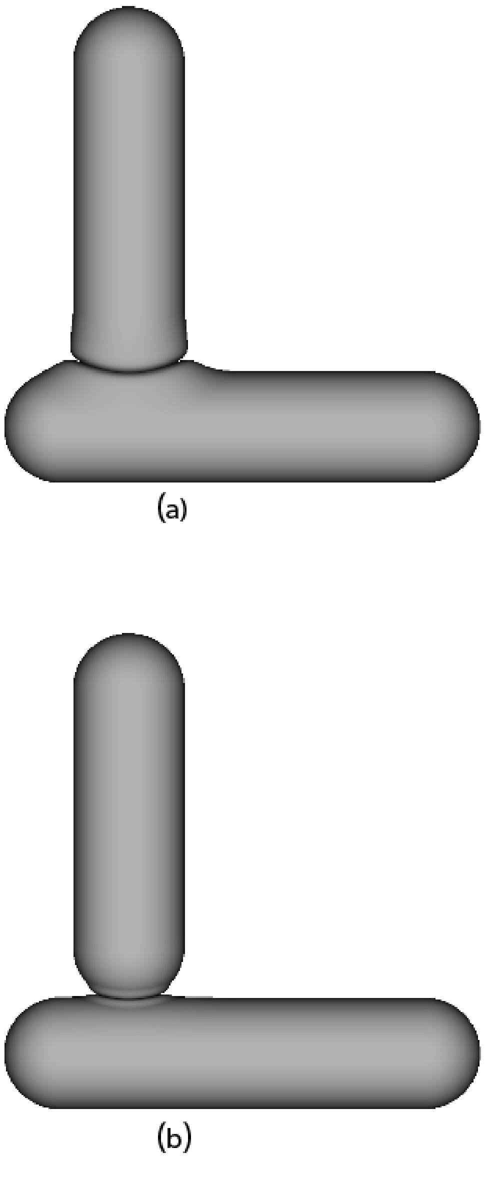


Figure 10: Resulting deformation without scaling of parameter h (a) and with scaling of parameter h (b).

The deformation term in this technique has three parameters to help control the deformation. Parameter c_j controls the width of the deformation. The lower c_j the wider the deformation becomes. Parameter m controls where the top of the deformation should be, and parameter h controls how large the deformation should be. Figure 11 at the end of the paper, shows comparisons of deformations with different parameters. The figures illustrate how each of the parameters effect the deformation. As default values $c_j = 0.4$, $m = 1.3$ and $h = 0.5$ have been used. These values gave nice results, with not too much deformation and both the width and center of the deformation seemed natural. Figure 12 shows resulting rendering using this technique.

5.4 Performance

The implementation of the improved single layered technique have been tested with a 2.80GHz Intel Core i5 CPU, 8GB Memory, and a NVIDIA GeForce GTX 570 with 1280MB memory GPU. Results from testing can be seen in the table below. We have used frames per second(FPS) to measure performance.

Resolution	No. Points	No. Lines	FPS
256x256	10	0	47
256x256	5	2	23
256x256	40	0	17
256x256	50	0	16
256x256	4	3	17
512x512	10	0	24
512x512	5	2	10
512x512	40	0	7
512x512	50	0	6
512x512	4	3	7

Using this technique with only point-skeletons performs quite well. However, adding line-skeletons slows it down fast. In any case, the results are promising. The implementation can be optimized in several ways, for example by adding bounding boxes to the objects and by implementing adaptive step size in the raycasting algorithm.

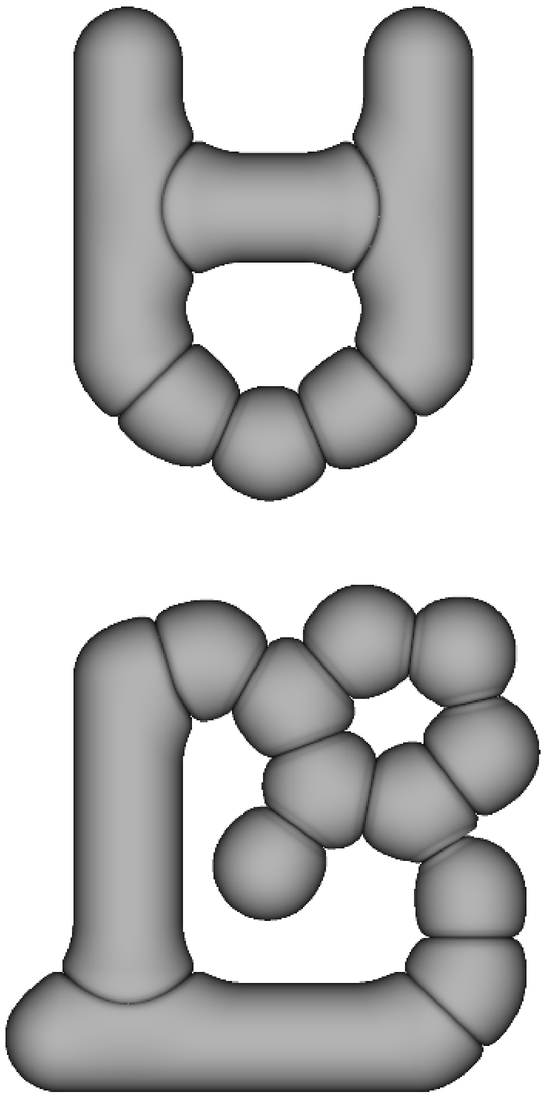


Figure 12: Resulting rendering using our technique. Default parameters.

6 Conclusions and Future Work

In concusion, we have implemented a working environment for rendering imlicit skeleton based objects and the deformation of these when they collide. The environment is interactive for both point and line skeletons. This is without any bounding boxes or any other optimizations. For future work this could be implemented to give better performance. The ray casting algorithm can be improved by implementing adaptive step size. Currently the algorithm use the same step size all the time. The step size can be varied by looking at the distance to the closest object. If the step size is set to the distance to the closest object, we avoid a lot of calculations while we are stil sure that we don't miss anything. In addition future work could look at the intersection of deformations. Currently the implementation does not check for any intersection of deformed objects, but only the original objects. This means that if the deformed sections of two objects intersect, the implementation does nothing to deform these objects further or alter the deformations in any way.

A natural extension of this work is to add physical forces to the objects. The focus of this paper was only on the visual part and adding forces was outside the scope of this work. For this paper we have used distance functions to model the objects. For future work it would be nice to have deformations working with convolution surfaces as well so it could be combined with the work from the previous project mentioned in the introduction. Another thing to consider is deformation from deformations.

As of now only the original objects are taken into account when calculating deformations.

References

[1] Alexis Angelidis. Adaptive implicit modeling using subdivision curves and surfaces as skeletons. *seventh ACM symposium on Solid modeling*, 2002.

[2] Alexis Angelidis, Pauline Jepp, and Marie-Paule Cani. Implicit modeling with skeleton curves: Controlled blending in contact situations. *nternational Conference on Shape Modeling and Applications (SMI'02)*, pages 137–144, 2002.

[3] MP Cani. Subdivision-curve primitives: a new solution for interactive implicit modeling. *Shape Modeling and Applications, SMI*, 2001.

[4] M.P. Gascuel. An implicit formulation for precise contact modeling between flexible solids. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 313–320. ACM, 1993.

[5] Agata Opalach and Marie-Paule Cani. Local deformations for animation of implicit surfaces. *13th Spring Conference on Computer*, pages 1–9, 1997.

[6] Jag Mohan Singh and P J Narayanan. Real-time ray tracing of implicit surfaces on the GPU. *IEEE transactions on visualization and computer graphics*, 16(2):261–72.

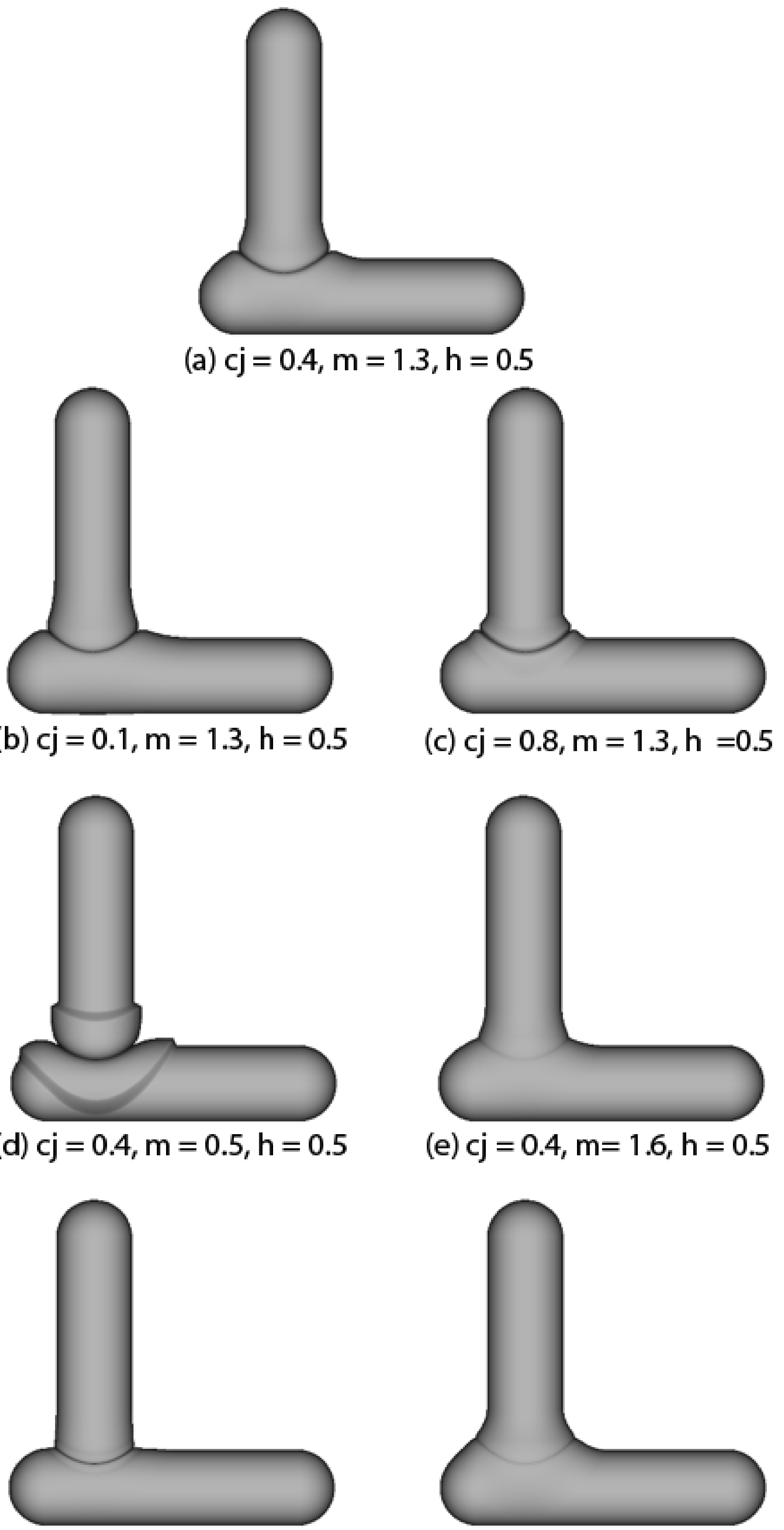

(a) cj = 0.4, m = 1.3, h = 0.5
(b) cj = 0.1, m = 1.3, h = 0.5
(c) cj = 0.8, m = 1.3, h =0.5
(d) cj = 0.4, m = 0.5, h = 0.5
(e) cj = 0.4, m= 1.6, h = 0.5
(f) cj = 0.4, m = 1.3, h = 0.1 0.
(g) cj = 0.4, m = 1.3, h = 0.7

Figure 11: Deformations using our technique with different parameters