Interactive Shape-Aware Deformation of 3D Furniture Models

Lea Aichner*
Supervised by: Associate Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer†

Institute of Computer Graphics
Technical University Vienna
Vienna / Austria

Abstract

Resizing of 3D models can be very useful when creating new models or when reusing old ones. However, naive resizing can create serious visual artifacts which destroy the characteristics of an object. In this work an algorithm that protects the features of 3D models during resizing is introduced. It is specialized for furniture models because it should be applied to a furniture configurator. We observed that the distortion that occurs during scaling is not distributed uniformly across the object. Our algorithm automatically detects the vulnerable parts of a model and then stretches only the non-vulnerable ones. Furthermore, the algorithm takes into account that when scaling a mesh in a specific direction, the texture has to be adapted as well in order to prevent representation errors.

Keywords: Deformation, Distortion, Mesh, 3D Model, Resizing, Scaling, Structure, Texture

1 Introduction

Starting from the late 1990s, digital geometry has emerged as a new type of digital media. Discrete digital models such as meshes are widely used in many applications, for example in entertainment, design, or engineering. They allow for a great flexibility regarding modifications and adjustment. Generating such models is still time-consuming and demands a lot of experience. Because of this, there is an emerging trend towards the reuse of existing models, parts, or designs [6].

In practice, models often have the same structure but a different scaling. A natural idea in such scenarios is to reuse the existing models by reshaping them, for example through resizing. The simplest approach to resize a geometry is to apply a global scale to the mesh along a specific direction. However, this will lead to unwanted distortions of significant features. An example of the distortion of a mesh and the texture can be seen in Figure 1 (c).

Clearly, this visual distortion depends on the magnitude of the scale and is not distributed uniformly across the surface. The visual artifacts are located in specific, vulnerable regions on the surface. Other regions remain visually correct regardless of the magnitude of the scaling. This observation shows that it is important to resize a mesh non-homogeneously, protecting some parts, while stretching others excessively.

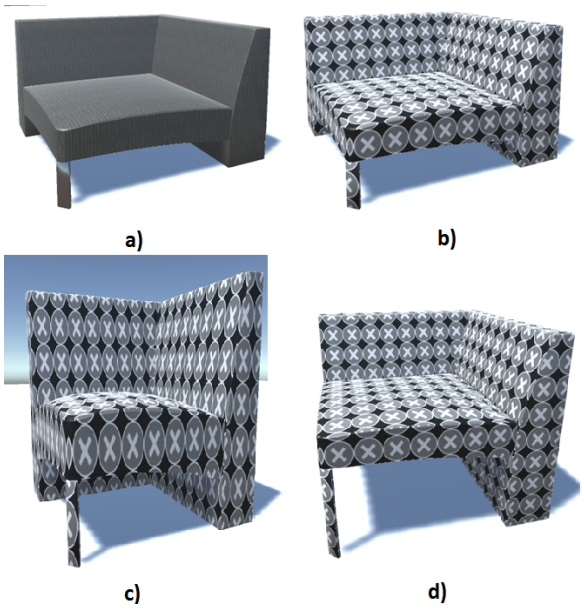

Figure 1: The Figure (a) shows the original furniture model. (b) shows the model with a test texture. (c) Applying a global scale leads to unwanted distortions at the seat, the back and the texture. (d) Scaling the model with our method preserves the characteristic features and corrects the texture.)

Our goal is to implement an algorithm that enables stretching a model without destroying its characteristic features. For example, rounded edges should not be distorted when scaling an objects along a specific direction (Figure 1 (d)) . Therefore, our algorithm takes into account the structure of the 3D model and avoids visual artifacts when scaling at runtime. The possibilities to extract such high-level information in digital models are still limited. Instead, our algorithm uses low-level analysis of the geometry to automatically detect vulnerable parts given predefined scaling axes.

In contrast to already existing context aware scaling al-

*lea.aichner@yahoo.de
†wimmer@cg.tuwien.ac.at

gorithms, our work discusses the adjustment of the texture after scaling as well. Even if the algorithm scales only the regions that are indifferent to the scale, the texture has to be adapted on the whole 3D model (Figure 1 (d)).

Our approach is specialized for parts of furniture models (Figure 1) because it is supposed to be applied to a furniture configurator. The models are characterized by rather simple geometries which have a regular shape along a specific scale direction. However, with some modifications it can be applied to complex models as well.

In order to facilitate the integration into existing Unity projects, the algorithm is implemented using the game engine Unity.

The next Section, Related Work, describes the state of the art of Shape-Aware deformation techniques. The third Section, Structure Aware Resizing Algorithm Overview, gives an overview of our algorithm and discusses its practical relevance. The Section Implementation describes the analysis phase and the processing phase of our algorithm. The fourth Section, Results and Evaluation, contains the results of evaluation achieved by our implementation. The last Section, Conclusion and Future Work, includes a discussion of possible improvements and extensions to our application.

2 Related Work

In this section we present some algorithms that are used to edit existing shapes. One approach is to work with free-form deformations, a space based deformation technique. Simple free-form deformations allow low-level mesh operations and provide a high degree of flexibility. To avoid destroying the whole structure of the object however, several methods have been proposed to perform higher-level deformations. The free-form deformations implemented by Sederberg et al. [10] and Coquillart [1] use a low-dimensional, band-limited, volumetric basis to impose smooth, low-frequency deformations to the geometry. The goal of the algorithm is to preserve parts with high-frequency details, while the low-frequency parts should be deformed.

The drawback of free-form deformations is that they only have a local and non-adaptive way of preserving structure. They do not take into account the content of the shape or global relations.

In order to deal with structure-aware deformation Zheng et al. [12] uses global relations between parts of an object. Semantic parts that belong together are represented by object-aligned bounding boxes or shape components obtained from segmentation. When deforming an object, Euclidean invariants (symmetries) are used to propagate edits to affect all symmetric elements similarly.

A similar idea is used by iWires [2] (Figure 2). The iWIRES framework of Gal et al. [2] uses global relations to achieve structure-aware deformation. The method is based on the researches of Singh et al. [11] and Orzan et al. [8], which show that an entire shape can be defined by a small set of curves. Furthermore, Singh et al. [11] defined the name wires to denote the curves that are key structural features capturing the shape.

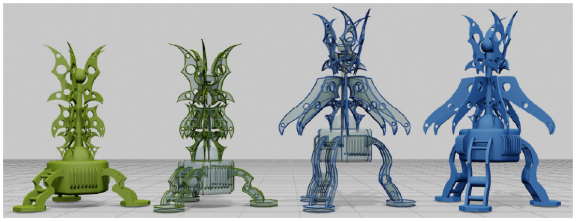

Figure 2: A complex model (left) that consists of 108 different components is analyzed and 250 intelligent wires (in green) are extracted. Editing a few wires leads to a new wire configuration (in blue). The result is shown on the right side. (Image taken from Gal et al. [2].)

In order to get the wires, the algorithm has to extract feature curves. Doing that in general models is challenging, but for models depicting man-made objects, sharp crease lines are good candidates [7]. Single wires can be combined to form a group that also holds geometric characteristics and information about relations.

The user has the possibility to deform the model using predefined handles. The algorithm propagates the deformation to the closest wire and enforces its individual characteristics. Thereby, it is important to maintain the group characteristics and group relations. An example of the iWires framework can be seen in Figure 2. The algorithm extracts 250 intelligent wires of the model, shown in green. Editing a few wires leads to a new wire configuration and can be seen in blue on the right side of the figure.

The algorithm of Kraevoy et al. [6] allows to stretch 3D objects along several directions, while protecting the model features and structures during resizing. To avoid distortion in particular parts of the object (Figure 3 (b)) it is important to scale the object non-homogeneously. For instance, while the proportions between the dial and the pendulum of the clock in Figure 3 (c) change, the model still appears visually correct. The algorithm detects all vulnerable regions in the model and records this information in a protective grid defined around the object, the so called *vulnerability map* (Figure 3 (d) and (e)). The digital model is then scaled non-homogeneously by a space-deformation technique with respect to the vulnerabiliy map. The computation of the vulnerability according to a specific scale axis is based on two components, *slippage* and *normal curvature*.

The slippage analysis [3] specifies how persistent a surface is to a given transformation or transformation type. It measures if a local region on a surface remains on the surface after the transformation is applied. This is the case when the surface normal of the local region is perpendicular to the scale axis. The normal curvature of the surface in the scaling direction predicts the amount of surface

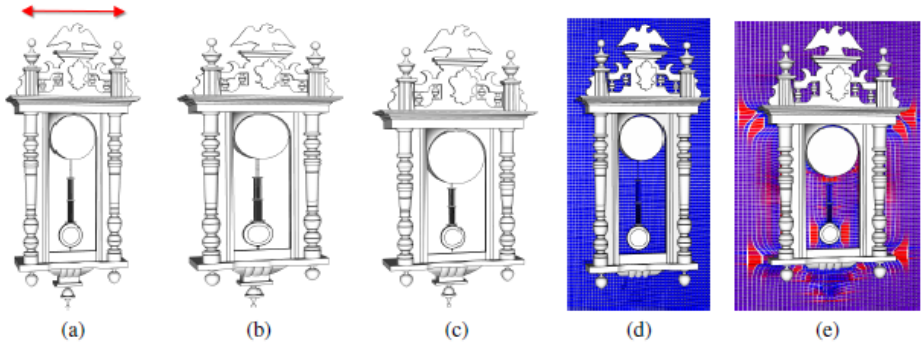

Figure 3: Resizing a clock model. (a) Standard non-uniform scale distorts the shape of parts of the model, e.g. the dial (b). Non-homogeneous Resizing resizes the clock in a more natural manner protecting its shape (c). Images (d) and (e) show part of the protective grid before and after resizing. (Image taken from Kraevoy et al. [6].)

bending subject to the scale. The method measures normal curvature at mesh vertices, projecting the scale axis to the surface tangent plane.

After the computation of the vulnerability map, the algorithm computes the scale gradients for each cell and for each direction. The aggregation of the scale gradients of each cell results in the global resizing transformation.

Linear blending methods like the methods of Schaefer et al. [9] or Kavan et al. [5] are often used in interactive space deformation because of their speed: each point on the object is transformed by a linear combination of a small number of affine transformations.

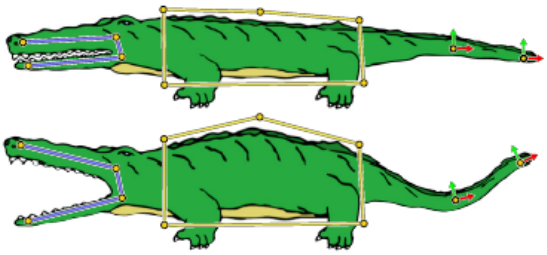

Figure 4: Bounded Biharmonic Weights for Real-Time Deformation support points, bones and cages as deformation handles. (Image taken from Jacobson et al. [4].)

Jacobson et al. [4] specifies an approach for real-time deformation of arbitrary 2D and 3D shapes by supplying weights for a linear blending scheme that produces smooth and intuitive deformation for handles of arbitrary topology (Figure 4). The system supports points, bones and cages as deformation handles. Bones are used to control rigid parts, cages to enlarge areas and points to transform flexible parts. The handles have maximum influence in their immediate environment.

3 Structure Aware Resizing Algorithm Overview

Our structure aware resizing algorithm will be used in an interactive 3D furniture configurator. The user gets the ability to create individual furniture such as a sofa. Furthermore, the user can configure a lot of properties of the furniture, for example the texture, the seat height, or the width of the arm rest. Because of the fact that the user can modify every single component of the furniture, including armrest or feet, a 3D representation of these single components is needed.

Unfortunately, this leads to a big amount of data and redundant information. For example, if the user can choose among ten different forms of arm rests and four different arm rest widths, the application has to save forty different 3D meshes. It would obviously be better to save only one mesh for every form and adjust its width at run-time. In the following sections, a content preserving algorithm that aims to solve this problem is presented. Furthermore, this algorithm takes into account that, when scaling a mesh in a specific direction, the uv coordinates have to be adapted as well in order to prevent representation errors.

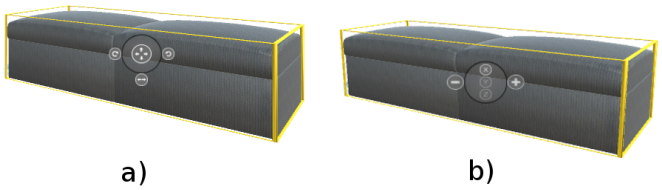

Figure 5: The Figure (a) shows the UI menu for rotating and translating the element. When clicking on the button on the bottom the scaling menu appears, shown in Figure (b). The bounding boxes are rendered in yellow.

The algorithm is implemented in Unity, a cross-platform game engine developed by Unity Technologies. The whole application can be divided into four sections:

1. The visualization of the furniture geometry: A selected element has to be highlighted and a user interface is needed to manipulate the element (Figure 5).

2. The processing of the user input: The user needs the ability to navigate through the scene, to select elements, to rotate, to translate and of course to scale them (Figure 5).

3. The *Analysis Phase* of the algorithm: In a preprocessing step the areas of the model that can be scaled along a specific axis are computed. Furthermore, the uv coordinates are analyzed in order to simplify their manipulation.

4. The *Processing Phase* of the algorithm: This step is executed at run-time. The world coordinates and texture coordinates are updated according to the results of the analysis phase.

The last two steps, Analysis Phase and Processing Phase, are described in detail in the following sections.

4 Implementation

The main idea of our algorithm is to detect the parts of a model that can be scaled along a predefined axis without destroying its salient features and adapting the texture coordinates to prevent representation errors.

Our algorithm is divided into an analysis phase and a processing phase. In the first part of the analysis phase, the algorithm analyzes the projection of the triangle normal onto the scaling axis and determines the parts of a mesh that can be scaled. However, the adjustment of the world coordinates leads to a distortion of the texture of the model. In the analysis phase of the texture coordinates the algorithm first computes the mapping of the texture on the 3D model. The next step is to detect all vertices of the model, whose texture coordinates have to be adjusted when scaling it along a specific axis. Furthermore, the scale direction and the correlation between the texture coordinate and world coordinate of a vertex are calculated. In the processing phase all world coordinates and texture coordinates are updated at runtime, based on the results of the analysis phase.

4.1 Analysis World Coordinates

Similar to the algorithm of Kraevoy et al. [6], our algorithm uses the projection of the triangle normal onto the scaling axis to decide whether a part of an element can be scaled or not.

First, the algorithm runs through all triangles in a furniture mesh. For each triangle the projection of the normalized surface normal onto the scaling direction is computed. This results in a triplet of scalars γ_x, γ_y and γ_z that indicate how vulnerable the triangle is to a scaling along the x,y and z axis, respectively. The surface normal is perpendicular to the scale axis if the projection γ onto the axis is zero. In that case, this part of the geometry can be scaled without destroying content features. Obviously, that implicates that all triangles with a projection γ unequal to zero shall not be adjusted. Due to imprecise vertex values, the projection γ is seldom exactly zero, even if the surface is parallel to the scale axis. Therefore, the application uses a *tolerance-threshold* ε (for example $\varepsilon = 0.02$). If the absolute value of the dot product is bigger than ε, this part of the geometry must not be changed.

For the sake of simplicity, the algorithm first computes all parts of the mesh that should not be modified. In order to specify these parts, the algorithm uses *ranges*. One range is defined by only two values indicating the start and the end of the range. If for example the scalar of the normal and the x axis is bigger than the tolerance-threshold ($\gamma_x > \varepsilon$), the minimum $\min(v_{1x}\ v_{2x}\ v_{3x})$ and the maximum $\max(v_{1x}\ v_{2x}\ v_{3x})$ of the vertices generate a new range r_x. Before saving the range, the algorithm has to detect if the new range r_x overlaps with an already existing range. If this is the case, the two ranges are merged into one. The tolerance-threshold can be modified by the user at runtime.

After running through all the triangles, the program can calculate the *Scale Rectangles* based on the non-scalable ranges. A Scale Rectangle is a 3D box that surrounds the 3D model and contains all vertices that can be scaled. The Scale Rectangles for different scale directions with different tolerance-thresholds ε can be seen in Figure 6. One Scale Rectangle is defined by two values SR_{min} and SR_{max} indicating the start and the end of the Scale Rectangle and a factor SR_{ratio} that describes the proportion of the size compared to the other Scale Rectangles. If a Scale Rectangle is bigger than another, the bigger one has to be stretched more than the smaller one in order to maintain the correct proportions of the model.

4.2 Analysis Texture Coordinates

The analysis phase of the texture coordinates is more complex than the analysis phase of the world coordinates. Therefore, we divide it into 3 steps.

4.2.1 Scale Factor Calculation

First, the algorithm has to find out the mapping of the texture coordinates if it is not known by default. The mapping specifies which distance in world coordinates corresponds to the distance of 1 in the texture coordinates.

The precondition for a correct calculation of the mapping is that the used textures have the same mapping in the u and v direction because our algorithm takes into account only the y coordinates of the geometry. A distance of 1 in the u-coordinates corresponds to the same distance in the world coordinates as a distance of 1 in the v-coordinates.

The calculation of the mapping is performed while running though all mesh triangles during the analysis phase of

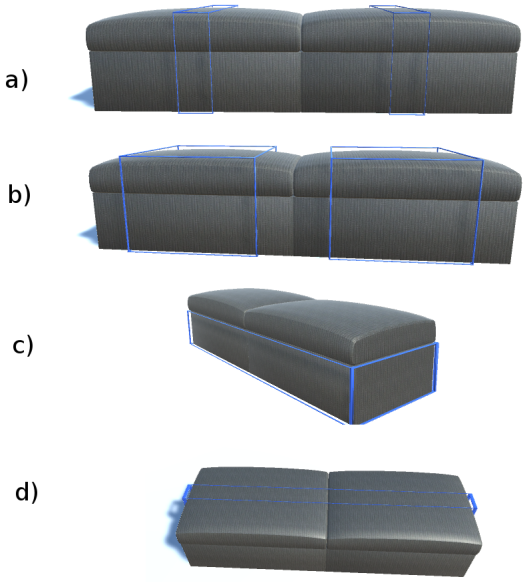

Figure 6: The figure displays the Scale Rectangles for the different scale axes. (a) The Scale Rectangles for the scale axis x with the default threshold $\varepsilon = 0.02$. (b) The Scale Rectangle for the scale axis x with the threshold $\varepsilon = 0.055$. (c) The Scale Rectangle for the scale axis y. (d) The Scale Rectangle for the scale axis z.

the world coordinates. One triangle consists of three vertices with y-world coordinates y_i, $i = 1,2,3$ and the corresponding texture coordinate v_i, $i = 1,2,3$. For the computation of the scale factor we are only referring to the v-coordinate of the UV coordinate system. First, the algorithm searches the two vertices with the largest distance $d_{i,j}$ in the y-world coordinates (Equation 1).

$$d_{i,j} = max(|y_i - y_j|), \quad i,j = 1,2,3. \tag{1}$$

Afterwards, it computes the local scale factor sF_l by dividing the distance in the texture coordinates of the vertices i and j by the distance $d_{i,j}$ (Equation 2).

$$sF_l = |v_i - v_j|/d_{i,j}. \tag{2}$$

In the end, the algorithm sets the local scale factor sF_l computed by the triangle with the largest distance $d_{i,j}$ as the final scale factor sF.

4.2.2 Texture Coordinates Identificaton

Let us assume that $V \subset \Re^3$ consists of all vertices of a 3D model. When scaling the model in the x direction, V_x holds all vertices whose world coordinates have to be adjusted. As shown in the Figure 7, not all texture coordinates in V_x have to be modified. The texture coordinates of perpendicular parts to the scale axis must not be adapted, in the picture highlighted with a red color. All triangles of the model whose uv coordinates need to be adjusted are identified by T_x.

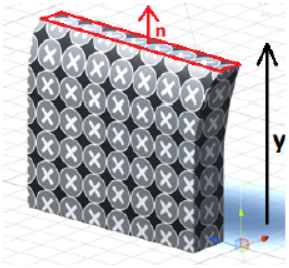

Figure 7: The texture coordinates of perpendicular parts to the scale axis must not be adapted, in the picture highlighted with a red color.

When running through the mesh triangles, the program saves the index of all vertices whose normal is smaller than a certain threshold λ (λ is by default 0.8). These triangles are not perpendicular to the scale axis and therefore their texture coordinates need to be adjusted. The threshold λ is used to make the algorithm more robust against inaccuracies of the model and can be changed by the user if needed.

4.2.3 Scale Direction and Correlation Identification

When changing the x coordinate of a vertex, the question is whether to modify the u or the v coordinate of the UV coordinate. Therefore, the algorithm has to determine the so called *UV Scale Axis* for every vertex in T_i.

Furthermore, an enlargement of a world coordinate does not lead automatically to an enlargement of the texture coordinate. The application needs to describe the correlation between world coordinates and texture coordinates for every single vertex.

The algorithm calculates the UV Scale Axis for every triangle in T_x, T_y and T_z using the previously computed UV Scale Factor. In order to compute the UV Scale Axis, the algorithm runs though all triangles T, whose texture coordinates have to be adjusted. Assuming the scale direction is the x-axis, then the algorithm first computes the distance $d_{i,j}, \quad i,j = 1,2,3$ between all x coordinates of the vertices that compose the triangle. The distance is then multiplied by the UV Scale Factor sF. The result is the distance in uv coordinates that the vertices should have $\widetilde{d_{uv}}$. In the next step the correct distance for both the u coordinates and the v coordinates are computed based on the original uv coordinates of the mesh (d_u and d_v). The algorithm compares then the correct distance d_u and d_v with the calculated distance $\widetilde{d_{uv}}$. If $\widetilde{d_{uv}} \approx d_u$, an enlargement in the x direction leads to an adaption of the u coordinate of the texture. If $\widetilde{d_{uv}} \approx d_v$, the v coordinate has to be corrected.

Furthermore, the correlation between the texture coordinate and world coordinate of a vertex is calculated by comparing the sign of the correct distance d_u or d_v and the calculated distance $\widetilde{d_{uv}}$. If they are the same, the vertex has a positive correlation. If they are different, an enlargement

of the world coordinate means a reduction of the texture coordinate. Obviously, the same vertex is processed multiple times because it is part of several triangles. In order to save the most significant value, our method saves the results calculated from the biggest triangle.

4.3 Processing Phase

After the analysis phase the 3D model has to be scaled at run-time. This scaling process is again divided into the adjustment of the world coordinates and the adjustment of the texture coordinates.

When scaling the element for example in the y direction, the algorithm runs though all previously computed Scale Rectangles. All vertices that have a bigger y coordinate than the minimum value of the Scale Rectangle SR_{min} have to be adapted. If the 3D mesh consists of n Scale Rectangles and it is scaled along the y axis by the amound of d, the algorithm starts with the Scale Rectangle with the minimal SR_{min}. The y coordinate of all vertices with a bigger y coordinate than SR_{min} are shifted by the amount of $SR_{ratio} * d$. The SR_{min} value of all other vertices has to be adjusted too. Then, the algorithm continues these steps with the next Scale Rectangle.

After adjusting the world coordinates for one Scale Rectangle, the uv coordinates have to be adapted too. If the world coordinates of a vertex have changed by the amount of d and if the texture coordinate of the vertex has to be adjusted, the uv coordinate of the vertex is adapted depending on the above computed UV Scale Factor sF, UV Scale Axis and correlation. That means for a positive correlation and a UV Scale Axis 'v' $v_{new} = v_{old} + sF * d$ and for a negative correlation and a UV Scale Axis 'u' $u_{new} = u_{old} - sF * d$.

5 Results and Evaluation

Most of the geometries we used for testing will be used in the 3D furniture generator too. However, in order to identify the flexibility of the algorithm we also tested it with more complex 3D models.

5.1 Scale Rectangles and Scaling

In Figure 6 the Scale Rectangles for the three different scaling directions x, y and z are displayed. The images (a) and (b) display the Scale Rectangle for the x direction with different ε values. The smaller the value, the more accurate the Scale Rectangles. Furthermore, the figure shows that the algorithm is able to detect more than just one Scale Rectangle per scale axis.

Figure 8 shows the stool model before and after scaling it along the y direction. Comparing the wireframes in Figure (a) and (b) it can be seen that all vertices in the upper part of the model that are not inside the Scale Rectangle are not modified.

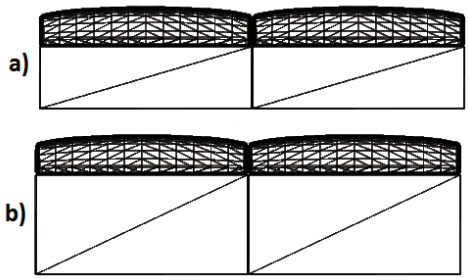

Figure 8: The figure shows the wireframe of a stool before (a) and after (b) scaling it in the y direction.

5.2 Textures

In order to analyze the adjustment of the uv coordinates, we used a simple test texture, as shown in Figure 9. When enlarging the 3D model as well as when reducing its size, the texture coordinates were adjusted in a correct manner. The texture repeated itself without causing artifacts. Since the relation between world coordinates and uv coordinates were considered too, the texture was properly adjusted on the opposite side of the model as well.

As we already explained above, not all texture coordinates of the updated vertices have to be adjusted (Figure 7). Even if the vertices on top of the model have been shifted, the texture coordinates remain the same. This is because the triangles are perpendicular to the scale direction.

5.3 Complex Models

Even if it is not important for the practical use, we tested the algorithm with more complex geometries. As shown in Figure 10, we applied the method to a bench. The left side shows the original geometry with the Scale Rectangles for the z-axis. After scaling it up (Figure 10 (b)), the characteristic features such as the stake in the middle are preserved. Furthermore, it can be seen that the algorithm can handle multiple textures on one geometry.

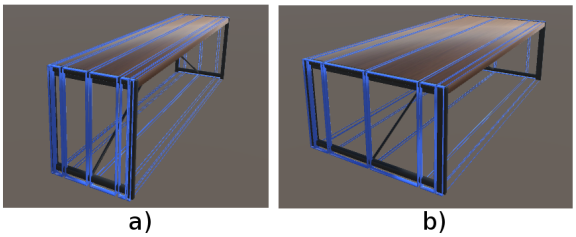

Figure 10: The figure shows the Scale Rectangles in z direction of a common bench (a) that is then scaled up (b). As can be seen the stake in the middle of the model has not changed.

It is important that the user has the ability to change the threshold ε manually. As shown in Figure 11 (a), the de-

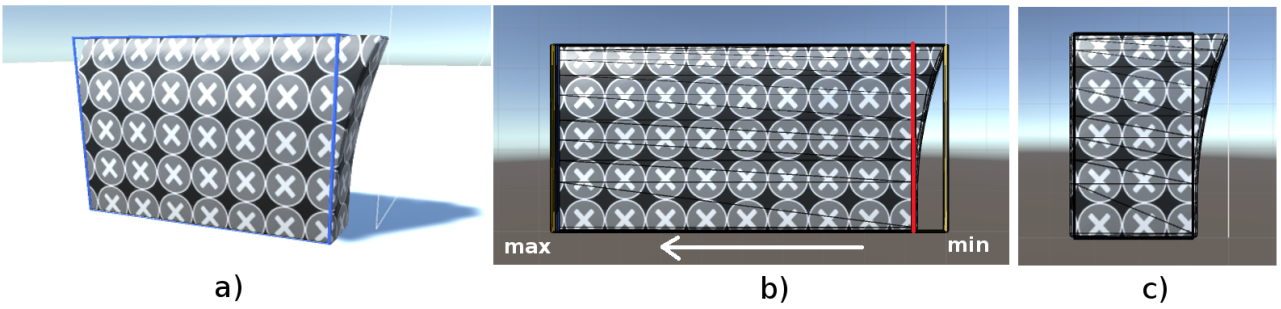

Figure 9: The 3D model of an arm rest with the test-texture. (a) Original model with the Scale Rectangle in z direction. Its size is 77.6 cm. (b) The armrest scaled to a size of 96 cm. The algorithm increased all vertices on the left side of the red line. (c) The armrest reduced to a size of 35 cm.

fault threshold is not enough if the lampshade should be scaled as well. In this case, increasing the threshold leads to more Scale Rectangles (Figure 11 (b)), which has an impact on the performance of the algorithm at run-time. Furthermore, we noticed that the more complex a geometry is, the more Scale Rectangles are computed.

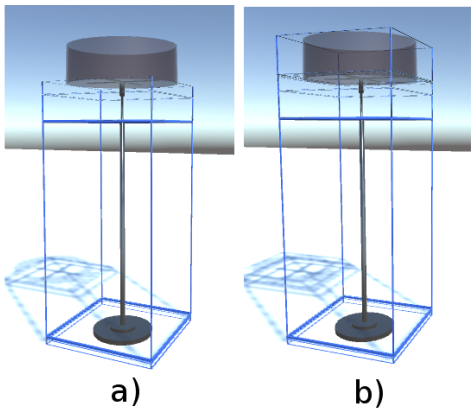

Figure 11: (a) The algorithm computes 6 Scale Rectangles when $\varepsilon = 0.02$. (b) When $\varepsilon = 0.03$ already 10 Scale Rectangles are computed.

The last geometry we used during the testing was a rather complex shelf (Figure 12). The parts of the shelf that contain no books were scaled extremely. By analyzing the section with the chemistry model, we noticed that the algorithm scaled it correctly and in an intuitive way. Only the foot of the model was scaled as well as the part of the books above it. The chemistry model itself remained the same. However, the geometry of the shelf shows the limits of our algorithm. Even if it scales the chemistry model correctly, the shelf is deformed in an unintuitive way.

6 Conclusion and Future Work

Resizing of 3D models can be very useful when creating new models. However, naive resizing can create serious

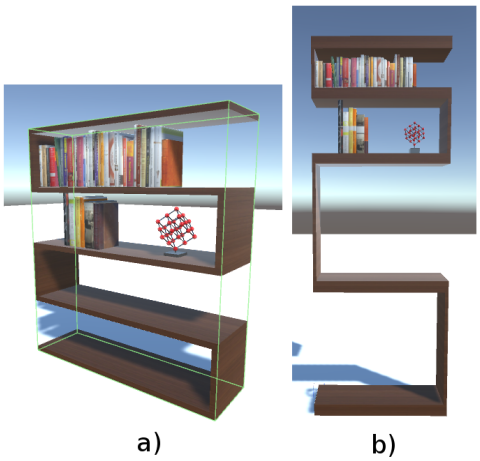

Figure 12: A complex geometry of a shelf. Even if our algorithm works correctly the shelf is not scaled in an intuitive way.

visual artifacts which destroy the characteristics of an object. This work presents a structure aware resizing algorithm that protects the model structures and takes the correct adaption of the texture into account as well. In order to facilitate the integration into existing projects, the algorithm was implemented using the game engine Unity. The application gives the user the ability to resize custom 3D models along predefined scaling directions at run time.

Our algorithm works well when working with rather simple 3D models that have a regular shape along a specific scale direction such as the arm rest, the stool, or the bench. Our method uses a threshold that indicates whether the world coordinates of a mesh triangle must be changed and a threshold that specifies whether a texture coordinate must be adjusted. Furthermore, the algorithm uses so-called Scale Rectangles to indicate all parts of the mesh that have to be scaled along a specific scale axis. For both thresholds a value can be defined that can be used over the entire geometry to compute the Scale Rectangles and to adjust the uv coordinates in a correct manner. Furthermore, only a small set of Scale Rectangles is computed.

This leads to a good performance when scaling the model at runtime.

However, our method has a number of limitations. When scaling complex geometries as the shelf in Figure 12 the algorithm computes unintuitive and unnatural results. This is because the algorithm computes too many Scale Rectangles. Complex geometries consist of many parts, where the mesh triangles are not parallel to the scaling axis and therefore vulnerable to the scale. However, some regions are parallel to the scaling axis and produce a Scale Rectangle even if these regions are very small. For every Scale Rectangle the algorithm repeats the resizing calculation and the adjustment of the uv coordinates. Therefore, a big number of Scale Rectangles quickly leads to a bad performance.

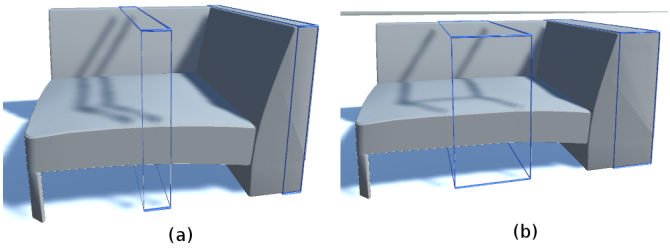

Figure 13: This piece of furniture is composed of several simple geometries. Nonetheless, the algorithm detects the correct Scale Rectangles.

In future work, we would give the user more possibilities to influence the calculation of the Scale Rectangles. When scaling the piece of furniture in Figure 13, one might prefer to scale only the seat and ignore the arm rest. Because of this, it would be reasonable to allow the user to delete selected Scale Rectangles after their computation or define new ones manually.

Furthermore, we would like to optimize the performance when scaling a 3D model. There are times when Scale Rectangles are so small that they have no visual impact on the results. Because of this, it would be reasonable to only allow Scale Rectangles bigger than a specific threshold. Alternatively, the algorithm could merge multiple Scale Rectangles that are close together into one.

Analyzing the state of the art leads to the conclusion that there is no general solution for scaling a model without destroying its geometric features. Depending on the structure of the geometry (man-made or not) and the field of application different algorithms have to be chosen. Nevertheless, because of its numerous benefits, structure-aware shape processing will remain a topic of research well into the future.

References

[1] Sabine Coquillart. *Extended free-form deformation: a sculpturing tool for 3D geometric modeling*, volume 24. ACM, 1990.

[2] Ran Gal, Olga Sorkine, Niloy J Mitra, and Daniel Cohen-Or. iwires: an analyze-and-edit approach to shape manipulation. In *ACM Transactions on Graphics (TOG)*, volume 28, page 33. ACM, 2009.

[3] Natasha Gelfand and Leonidas J Guibas. Shape segmentation using local slippage analysis. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 214–223. ACM, 2004.

[4] Alec Jacobson, Ilya Baran, Jovan Popovic, and Olga Sorkine. Bounded biharmonic weights for real-time deformation. *ACM Trans. Graph.*, 30(4):78, 2011.

[5] Ladislav Kavan, Steven Collins, Jiří Žára, and Carol O'Sullivan. Geometric skinning with approximate dual quaternion blending. *ACM Transactions on Graphics (TOG)*, 27(4):105, 2008.

[6] Vladislav Kraevoy, Alla Sheffer, Ariel Shamir, and Daniel Cohen-Or. Non-homogeneous resizing of complex models. *ACM Transactions on Graphics (TOG)*, 27(5):111, 2008.

[7] Yutaka Ohtake, Alexander Belyaev, and Hans-Peter Seidel. Ridge-valley lines on meshes via implicit surface fitting. *ACM transactions on graphics (TOG)*, 23(3):609–612, 2004.

[8] Alexandrina Orzan, Adrien Bousseau, Pascal Barla, Holger Winnemöller, Joëlle Thollot, and David Salesin. Diffusion curves: a vector representation for smooth-shaded images. *Communications of the ACM*, 56(7):101–108, 2013.

[9] Scott Schaefer, Travis McPhail, and Joe Warren. Image deformation using moving least squares. In *ACM transactions on graphics (TOG)*, volume 25, pages 533–540. ACM, 2006.

[10] Thomas W Sederberg and Scott R Parry. Free-form deformation of solid geometric models. *ACM SIGGRAPH computer graphics*, 20(4):151–160, 1986.

[11] Karan Singh and Eugene Fiume. Wires: a geometric deformation technique. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 405–414. ACM, 1998.

[12] Youyi Zheng, Hongbo Fu, Daniel Cohen-Or, Oscar Kin-Chung Au, and Chiew-Lan Tai. Component-wise controllers for structure-preserving shape manipulation. In *Computer Graphics Forum*, volume 30, pages 563–572. Wiley Online Library, 2011.