Repurpose 2D Character Animations for a VR Environment using BDH Shape Interpolation

Simone Barbieri[1,2,3], Ben Cawthorne[3], Zhidong Xiao[2], and Xiaosong Yang[2]

[1] Centre for Digital Entertainment, Bournemouth, United Kingdom
[2] Bournemouth University, Bournemouth, United Kingdom
[3] Thud Media, Cardiff, United Kingdom
`sbarbieri@bournemouth.ac.uk`

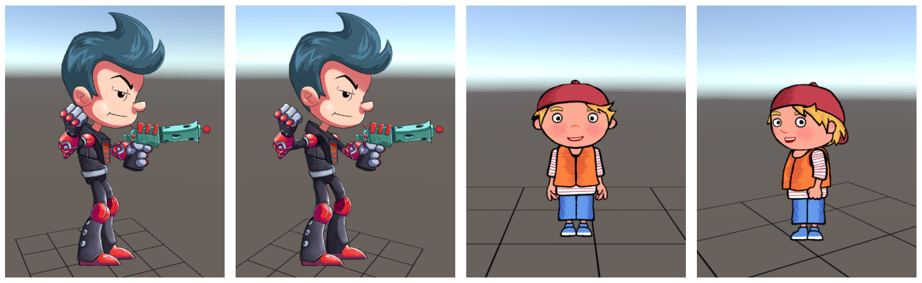


Fig. 1. Two examples of 2D characters placed in a 3D environment using our framework.

Abstract. Virtual Reality technology has spread rapidly in recent years. However, its growth risks ending soon due to the absence of quality content, except for few exceptions. We present an original framework that allows artists to use 2D characters and animations in a 3D Virtual Reality environment, in order to give an easier access to the production of content for the platform. In traditional platforms, 2D animation represents a more economic and immediate alternative to 3D. The challenge in adapting 2D characters to a 3D environment is to interpret the missing depth information. A 2D character is actually flat, so there is not any depth information, and every body part is at the same level of the others. We exploit mesh interpolation, billboarding and parallax scrolling to simulate the depth between each body segment of the character. We have developed a prototype of the system, and extensive tests with a 2D animation production show the effectiveness of our framework.

Keywords: Virtual reality · Animation · 2D characters in 3D environment · Shape interpolation · Billboarding · Parallax Scrolling

1 Introduction

In the last few years, the interest in Virtual Reality technology has greatly grown. The increase in the graphical processing power has, in fact, allowed consumers to get high-end experiences with a reasonable price. Despite the strong interest towards this technology, the quality content is still very limited at the moment, so the entire platform, at least for the entertainment industry, risks becoming merely a gimmick or a tech demo [28].

In the recent years, *indie games* have spread widely and have become an important reality in the video game industry [23]. Many of these games are realized by very few people, or, in some cases, even by single individuals, such as *Braid* [10] or *Undertale* [17]. Moreover, a large number of these games are realized in 2D. In traditional platforms, in fact, 2D represents a more economic and immediate alternative to the 3D animation, which is particularly diffuse in video games, but it is becoming increasingly popular in animation as well. As a matter of fact, while 2D animation requires as much skill as 3D animation, it is still faster to produce, since it has a dimension less to take in account while animating. The 2D asset production is also less expensive and faster to realize.

As Virtual Reality places the user in the centre of a 3D environment, many content creators that could bring quality products for the platform are stopped by the cost of the production of a full 3D environment and characters. Using 2D characters in VR would definitely reduce the production cost. However, there is an important challenge to solve in order to use 2D character in a 3D environment. 2D characters and animations, in fact, do not have any depth information. When they are inserted in a 3D context, without any revision, they would appear just flat, and they would ruin the user's immersion in the VR experience.

In this paper, we address this problem by presenting an original framework to retarget 2D animation contents for Virtual Reality application. We combine two classical computer graphics techniques – billboarding and parallax scrolling – with a new shape interpolation method to simulate the depth in 2D characters and animations, in order to exploit them in a 3D VR environment.

The problem we aim to solve with this paper could be considered as an extension of the problem to make appear and interact 2D object in a 3D environment. Depending on the medium, this problem is solved in different ways. For video games, for example, it is often used the previously cited billboarding, which rotates the 2D elements in the environment towards the camera, in order to do not let the user see that they are actually 2D. This technique is used especially with user interface, but even for effects, background elements – trees, clouds – or characters. This happens, for example in the video game *Mario Kart 64* [22], where the characters are prerendered from 3D models, but in the actual game they appear as 2D sprites. For what concerns films, it is easier to handle the problem. The camera in a film is, in fact, controlled by the director and the

[4] Copyright ©, Esoteric Software
[5] Copyright ©, Toot Enterprises Limited
[6] Copyright ©, Magic Mall, Cloth Cat Animation

user will never have the chance to watch somewhere that has not been settled by the director himself. Thus, inserting 2D elements is a straightforward task, usually performed by visual effects compositors. An example could be seen in the film *Who Framed Roger Rabbit* [35], in which cartoon characters interact with the real world. For traditional animation, letting the character interact with 3D object requires a similar procedure. With VR, it must be considered that the user can watch in any direction at any time, and even slightly move inside a predefined area, hence a traditional composite cannot be realized.

The rest of the paper is organized as follows: in Section 2 we present a brief review of the related work; in Section 3 we give an exhaustive explanation of the system, including a detailed description of the three technologies we combined for our framework; in Section 4 we show and analyze the results we obtained with the proposed system; in Section 5 we draw our conclusions and explain the future works that could be done to improve the framework.

2 Related Work

Merging 2D and 3D In this paper, we present a method that combine together 2D characters in a 3D environment. In computer graphics there are several subfields that attempt a similar task, or, more generally, merging 2D and 3D elements. Due to the great amount of different subfield in this category, we introduce in this section only the groups we believe are the most close and relevant to our work.

One of these subfields is the hybrid animation [25], which combines 2D and 3D animation media. There are several examples of 2D films that include 3D objects. One of the most remarkable example is *The Iron Giant* [9], which features an entire 3D character, the Iron Giant itself, in a 2D animated film. In 2002, in the film *Spirit: Stallion of the Cimarron* [4], DreamWork Pictures revealed a technology to use 3D characters while the camera is significantly far away from them, to "take over" to a 2D animation when the camera gets closer [15]. In the same year, Walt Disney Pictures, in the film *Treasure Planet* [14], presents a hybrid character, namely a 2D character with 3D components.

Sỳkora and colleagues [29] introduce a method to allow users to specify depth inequalities in some sections of a 2D character, in order to generate a 2.5D popup. This technique has different objectives, such as enhancing the perception of depth in a 2D character, producing 3D-like shading or even stereoscopic images. This method, however, produces several artefacts due to the incorrect estimation of the contour thickness. Jain et al. [21] propose a technique for adding a 3D secondary motion – the motion of objects in response to the one from a primary character – to a 2D character exploiting physical effects. Usually this kind of effects are hand-animated and they are particularly time-consuming. Their method, however, integrate simulation methods to replicate cloth motion.

To generate 3D animation from 2D sketches is another technique which combine 2D and 3D together. Similarly to the work presented in this paper, these techniques have to solve a depth problem, although they have to infer the depth, which is absent in the drawing, while we have to simulate it in order to display correctly the character in a 3D environment. Davis and colleagues [16] are the first to introduce a specific method to pose articulated figures with sketches. Their tool requires as input a keyframe sequence sketched from the artist and the system will reconstruct all the possible 3D poses that match the input sketch and that have a valid ranking score, which is computed according to a set of heuristics. The 3D poses are ordered by their ranking score, and the user can select the pose that is the closest to his idea. However, this method has a few problems, such as the necessity of the user to specify the template of the skeleton through a configuration file and the manual annotation of the sketch. These problems are addressed by Mao et al. [24], who present a tool in which the user does not need any more to manually specify the template of the skeleton; in fact, now he can select one from a list categorised by gender, ethnicity and age. Then the user draws the stick figure, by using the thickness of a connection between two joints to express the depth of that body part. Jain et al. [20] propose a method to recreate a hand-drawn animation in a 3D environment. The generated animation consists of a reconstruction of motion capture poses, which match the user's animation, but does not follow exactly the animator's drawing. The main difference with the previous methods is that exploits a database to reconstructs the animation.

More recent works in this subfield allow posing character with very few lines. Guay and colleagues [18], for instance, take the concept of the "line of action", a technique used in drawing to grant a dynamic look to a character, and, while giving it a formal definition, they use it to pose a 3D character with a single stroke. Hahn et al. [19] introduce an original concept: the sketch abstraction. Fundamentally, it is a way to connect the character's rigging to the 2D input sketch. The abstraction is bound to some points in the mesh. Then, it establishes a correspondence between the abstraction and the input sketch, which can be computed with a straightforward in-order correspondence by considering the drawing direction, or by using the closest-point matching between the two sets of points. The final posing is therefore retrieved by solving an optimisation problem. Barbieri et al. [7] propose a method to automatically compute the sketch abstraction, thus removing a redundant task for the user in the previous method.

The final subfield of computer graphics related to our work we introduce is the crowd simulation, which is the reproduction of the movement of a vast number of character at the same time, typically in a 3D environment. One of the techniques used to display this amount of entities is the image-based rendering. Tecchia et al. [30] follow this approach, by using pre-generating impostors [27] – textured polygons that face the camera and replace more complex objects – rendered from different viewpoints. Depending on the position of the user, the most appropriate impostor is displayed. This method has the downside of requiring a huge amount of memory, as it requires a render of each character, from different perspectives and multiple frames for each of them. Aubel et al. [5] solve this problem by using dynamically generated impostors. In this way, no storage is used for impostors that are not active in the scene. In addition to the movement of the camera, for the generation of the images for the impostors they have to take in account the self-deformation too, as the character are not static objects. They solve the problem updating the impostor only if the distance between some pre-determined points in the character's skeleton changes significantly. Yang and colleagues [34] combine this two approaches by using both pre-generated snapshots and synthesizing new ones dynamically. To produce the new images, they use segments of the pre-generated one, avoiding thus the rendering of the geometric model multiple times.

2D Shape Interpolation The interpolation of planar shapes is a well-known problem in computer graphics and there exist many approaches in literature which address this problem. Besides offering an exhaustive review of the most relevant classical methods of mesh morphing, Alexa [2] provides a comprehensive explanation of the terminology and mathematical background required for the proper understanding of the problem.

The As-Rigid-As-Possible (ARAP) technique [3] aims to generate rigidity-preserving interpolations. By blending the interior of the shape rather than the boundaries, it creates locally least-distorting in-between shapes. It linearly blends the rotation and scaling components of the transformation for each pair of triangles from the source and target shapes, to consequently reconstruct the shape sequence. Despite this method offers a better control on local distortions compared to classic methods, it also produces artefacts if the source and target shapes present a large-scale deformation. To solve this problem, Choi et al. [13] and Baster et al. [8] used a different procedure to choose the rotation angles and thus guarantee coherence between adjacent triangles.

Xu et al. [33] related the rigid interpolation to the Poisson problem, thus offering a formal mathematical definition to the problem. However, this method is almost identical to [3]. The main difference is the weighting, which allow the result to not depend significantly on the tessellation of the mesh. Nonetheless, this method undergoes the same problem of [3], presenting distorted results whether large rotations are applied.

Weber and Gotsman [32] presented a method to produce smooth conformal mappings by blending the *angular factor* – the local orientation change induced by the mapping – on the shape's boundary. Chen et al. [11] extended their method by supporting the wider class of quasi-conformal mappings, in order to provide interpolations with a bounded amount of conformal distortion. Chien et al. [12] improved [11] presenting an algorithm two orders of magnitude faster. Moreover, the method from Chien et al. is meshless, thus provides results with increased smoothness.

3 2D Characters in a 3D Environment

The proposed framework allows the users to repurpose 2D characters and animations in a 3D, VR environment. Our system simulates the depth between the different body parts of the characters by moving them in the scene according

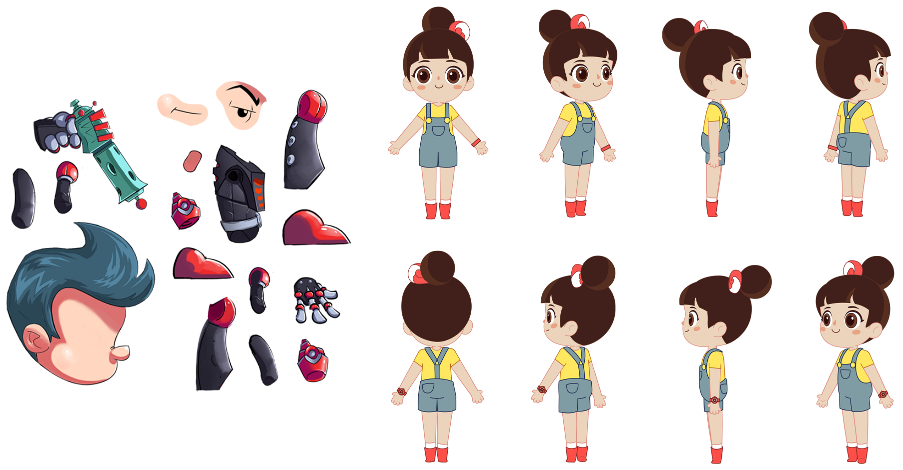


Fig. 2. The input of the system. Every character is split up into different body parts, as on the left[4], and it is provided drawn from eight different perspectives, as on the right[6].

to the position of the viewer. It exploits 3 different techniques: billboarding [1], parallax scrolling [6] and 2D shape interpolation [12]. As shown in figure 2, the system requires as input a character split in different body parts. Moreover, each body part has to be provided drawn from eight different perspectives.

The body parts are placed all on the same plane $\mathcal{P}$, the billboard, which will contantly rotate to always face the user. To determine the rotation matrix of the billboard which rotates around the y-axis, and with the viewer looking towards the negative z-axis, we first compute the eye vector from the model view matrix M:

$$\vec{V}_{eye} = M^{-1} \begin{pmatrix} 0 \\ 0 \\ -1 \\ 0 \end{pmatrix} \quad .$$

The rotation θ about the y-axis is then computed as:

$$cos\theta = \vec{V}_{eye} \cdot \vec{V}_{front} \quad ,$$
$$sin\theta = \vec{V}_{eye} \cdot \vec{V}_{right} \quad ,$$

where

$$\vec{V}_{front} = \begin{pmatrix} 0, 0, 1 \end{pmatrix} \quad ,$$
$$\vec{V}_{right} = \begin{pmatrix} 1, 0, 0 \end{pmatrix} \quad .$$

The rotation matrix R around the y-axis:

$$R = \begin{bmatrix} cos\theta & 0 & sin\theta \\ 0 & 1 & 0 \\ -sin\theta & 0 & cos\theta \end{bmatrix} \tag{1}$$

is then concatenated to the M matrix. The matrix MR is therefore used to transform the billboard geometry. A similar operation is used to rotate the billboard around the x-axis too, however the rotation is attenuated by an arbitrary value ξ.

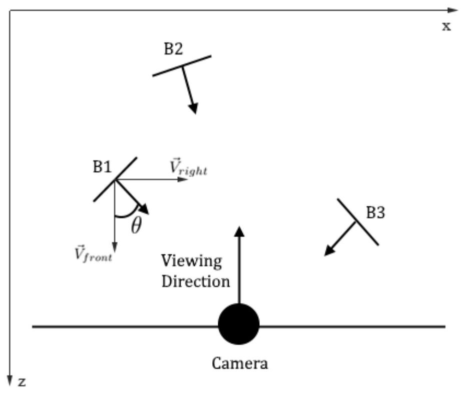


Fig. 3. This example shows how the billboarding works. B1, B2 and B3 are the billboards, which always face the camera.

On the billboard, the body parts move according to the movement of the user and to the parallax scrolling. When the viewer turns around the character, the body components in the closest half of the character's body move in the viewer's opposite direction, while the ones in the farthest half move in the same direction. The central part, such as the torso, differently, does not move at all, except for the rotation due to the billboarding. Furthermore, according to parallax scrolling, the closest parts move faster. The reversed movement of the farthest component is explained by the *negative speed* of their movement, as they move *slower* than the closest parts. A layer λ is assigned to each body part of the character that should move when the camera is rotated. λ represents the proximity of the body part with the user; a higher λ corresponds to a body part closer to the user. The torso's λ is 0, so it will not move. For the closest parts, it will be positive, while negative for the farthest ones. When the camera is moved by a translation T_{camera}, the body parts will simply be moved by:

$$T_{bodypart} = \frac{T_{camera} \cdot \lambda}{\zeta} \quad , \tag{2}$$

where ζ is a balancing parameter.

λ is computed automatically by the system. When the user provides the character, it is analyzed from 4 perspectives: front, left, right and rear. Based on the position of each component from each perspective, it will be assigned a different λ. We use the left and right perspective to compute the front and rear λ values, and viceversa. For each view point, the starting point $\lambda = 0$ is placed in the middle point, but it could be replaced by the user. The others λ are assigned on the opposite direction of the z-axis for the left and right perspectives, or the x-axis for the front and rear ones, at regular intervals. However, for the rear and left perspective, the sign of these values is changed. Figure 4 shows an example of this process just on the head of the character. In the figure, the $\lambda = 0$ position has been changed. The other values are then assigned in the opposite direction of the z-axis, so to the eyes, and the mouth will be assigned a λ value of 1, while to the chignon is assigned the value of -2. These values are used when the user is watching the character from a front perspective. In the rear perspective, the same values are used, but with the opposite sign. For the intermediate perspectives, front right, front left, rear right and rear left, we use the same λ values of the left and right ones.

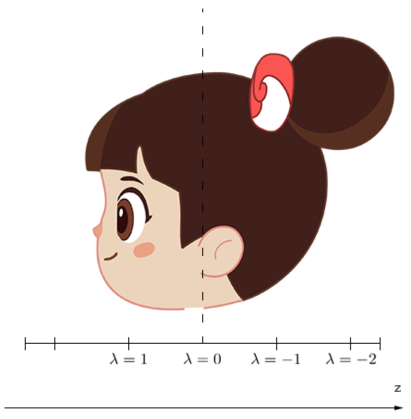


Fig. 4. An example of how the λ value is assigned to each component. In this example the left side of the character's head is taken into account. The reported values are not assigned to the body parts on the left side, but on the front one, and on the rear one with opposite sign.

As long as the user does not move significantly around the character, the shape interpolation is not applied. When the user surpasses a certain threshold, the perspective changes, and the interpolation is employed to change the appearance of each body part. Figure 5 shows the different perspectives angles and the thresholds.

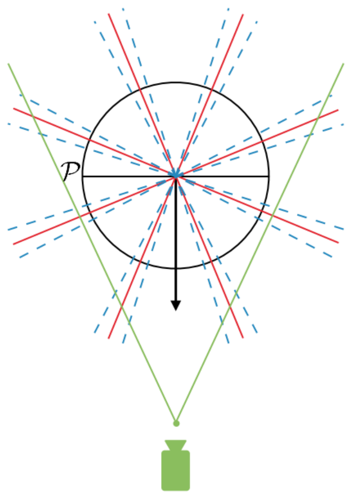


Fig. 5. The character from the top. The arrow is the viewing direction of the character. The red lines mark the points in which the perspective of the character changes. The dashed lines indicate the threshold for the interpolation.

The 2D shape interpolation is employed to produce a smooth transition during the change of perspective when the user turns around the character. Without it, the body parts would just sharply change shape whenever the user surpasses a certain angle. We divide the surface of the body part into two sections. The *joint part* – *jp* – is the section that appears in both the source and target surfaces. The rest is the *margin*. We call *appearing margin* – *am* – the one in the target mesh, while *disappearing margin* – *dm* – the one in the source. We highlight, however, that the difference between appearing and disappearing margin is based only on the walking direction of the user. Thus, they are not fixed as appearing or disappearing. The user is required to manually identify the joint part and the margin part of the mesh. Figure 6 shows an example of these sections.

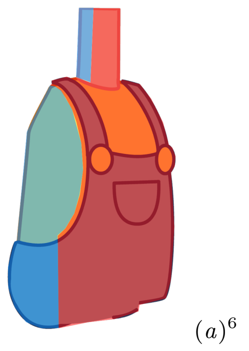
The interpolation is applied only if the user is between the two threshold. We define the interpolation as i^t, where $t \in [0, 1]$, $t = 0$ at the starting threshold and $t = 1$ at the ending threshold. Depending on the position of the user, a certain i^t is computed and shown.

Let us call a the angle between the character viewing direction and the vector from the character's position to the user's position, as illustrated by figure 7, t is simply computed from a and the two thresholds:

$$t := \frac{a - t_1}{t_2 - t_1} \quad , \tag{3}$$

where t_1 and t_2 are the angles of the first and second thresholds respectively.

Each interpolation i^t is a composition of two distinct operations. For what concerns the joint part of the shape, a classic shape interpolation method is ap-



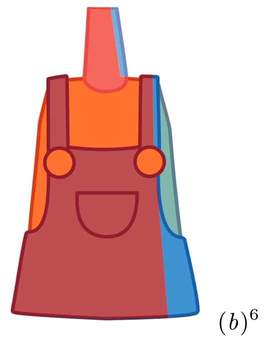


Fig. 6. The same body part from two different perspectives. The red section of these two body parts is the joint part. The blue parts are the margins.

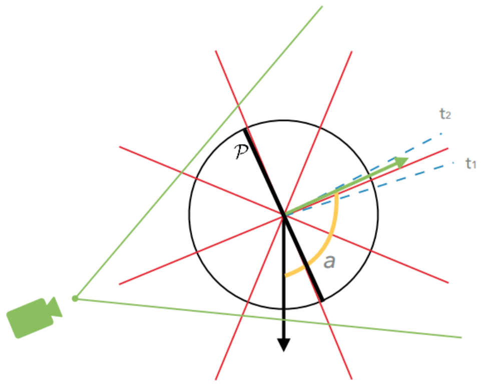


Fig. 7. In this example, the camera is between the two thresholds, and the angle a is computed for the correct interpolation.

plied. As we mentioned in Section 2, the 2D shape interpolation is particularly studied in computer graphics. There are several methods that solve this problem. For our system, we chose to rely on Bounded Distortion Harmonic (BDH) Shape Interpolation [12]. This method produce "provably good"[26] harmonic mappings – they are smooth, locally injective and have bounded conformal isometric distortion – such that its geometric distortion is bounded by the input mapping's one. Moreover, this method does not employ mathematical optimization. As our system requires to compute the interpolation for each body part of each character every time the user moves significantly in the scene, the parallel capability of this method makes it the most appropriate for our framework.

BDH shape interpolation allow us to obtain an interpolating function $f : [0,1] \times \Omega \rightarrow \mathbb{R}^2$ from two input locally injective sense-preserving harmonic mappings $f^0, f^1 : \Omega \rightarrow \mathbb{R}^2$. The idea of the method is to interpolate the Jacobian

J_f and then to integrate it to retrieve an interpolation. By identifying $\mathbb{R}^2$ with $\mathbb{C}$ and using the complex notation $z = x + \mathrm{i}y$, they define a planar harmonic mapping as a mapping $f : \Omega \to \mathbb{C}$ where $f(x + \mathrm{i}y) = u(x, y) + \mathrm{i}v(x, y)$ and the components u and v are harmonic. On a simply-connected domain Ω, any harmonic planar mapping f can be written as the sum of a holomorphic and an anti-holomorphic function:

$$f(z) = \Phi(z) + \overline{\Psi}(z) \ . \tag{4}$$

For such mapping, $f_z = \Phi'$ is holomorphic and $f_{\bar{z}} = \overline{\Psi'}$ is anti-holomorphic.

This decomposition allows them to interpolate the similarity and anti-similarity parts of J_f independently by interpolating f_z and $f_{\bar{z}}$, as they are holomorphic and anti-holomorphic respectively. Thus, they obtain Φ and $\overline{\Psi}$, the holomorphic and anti-holomorphic components of the interpolated mapping. These two components are defined by the Cauchy complex barycentric coordinates [31]:

$$\Phi(z) = \sum_{j=1}^{n} C_j(z)\phi_j, \quad \Psi(z) = \sum_{j=1}^{n} C_j(z)\psi_j \ . \tag{5}$$

To compute f_z, Chien et al. introduce a formula for f_z^t to linearly interpolate the angle of the closest rotation transformation by linearly interpolating the arguments of f_z^0 and f_z^1. This can be obtained by linearly interpolating the logarithms of f_z^0 and f_z^1, as expressed by this formula:

$$f_z^t = (f_z^0)^{1-t}(f_z^1)^t \ . \tag{6}$$

The following manipulation shows that these logarithms are linearly interpolated:

$$\begin{aligned} f_z^t &= \exp\Big((1-t)\log f_z^0\Big) \cdot \exp\Big(t \log f_z^1\Big) \\ &= \exp\Big((1-t)\log f_z^0 + t\log f_z^1\Big) \\ &= |f_z^0|^{1-t} \cdot |f_z^1|^t \cdot \exp\Big(\mathrm{i} \cdot (1-t)\arg(f_z^0) + t\arg(f_z^1)\Big) \ . \end{aligned}$$

f_z^t is essentially a holomorphic interpolation which linearly interpolates the argument. As such, it has to interpolate f_z^0 and f_z^1 so that the scaling constant is fixed.

For what concerns the anti-holomorphic part $f_{\bar{z}}$, Chien et al. introduce two different methods. Here we are going to discuss only the method we used for our framework. For $f_{\bar{z}}$ cannot be used the logarithmic interpolation, as $f_{\bar{z}}^1$ and $f_{\bar{z}}^1$ typically are 0 at points in Ω, and a logarithm for them cannot be defined.

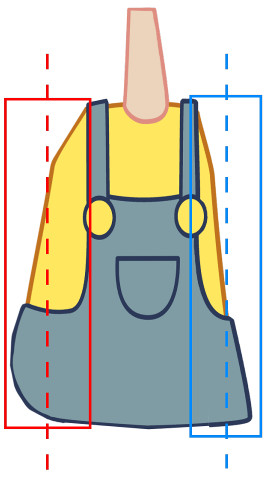
A new quantity is introduced: $\eta = g_{\bar{z}}\overline{g_z} = \mu|g_z|^2$, which is anti-holomorphic and where g is a planar mapping. This quantity is linearly interpolated by the following formula:

$$\eta^t = (1-t)\eta^0 + t\eta^1 \ , \tag{7}$$

Chien et al. also introduce a scaling function $\rho : [0,1] \to (0,1]$ such that $\rho(0) = \rho(1) = 1$, in order to check the geometric distortion bounds for each time t and scale η accordingly by preserving the bounds. We then obtain:

$$f^t_{\bar{z}} = \frac{\rho(t)\eta^t}{\overline{f^t_z}} \ . \tag{8}$$

Together, equations 6 and 8 give us the formulae to compute f^t_z and $f^t_{\bar{z}}$, which, integrated, allow us to obtain the interpolated mappings.


$(a)^6$

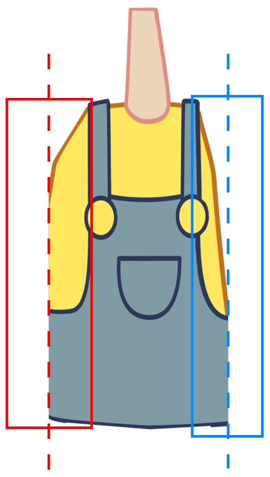

$(b)^6$

Fig. 8. An example of the interpolation of the margins at $t = 0.5$. The two different perspectives are those in figure 6. While the central part is the result of the deformation of the BDH interpolation, an intermediate state between the origin and target transformations, the other two sections, marked by the red and blue boxes, are the margins. As in this example we are considering $t = 0.5$, those sections are cut by half.

For what concerns the margins, let us consider the plane $\mathcal{P}$ on which lay the mesh of each body part. We already mentioned that the mesh is split in two different parts: the joint part and the margin. Let us indicate the width of the margin as m_{width}. For an interpolation i^t, we cut the plane $\mathcal{P}$ along the y-axis at $x = m_{width} \cdot t$. If the margin is an appearing margin, the vertices on the closest part of the margin to the joint part will be added to the mesh. Instead, if it is a disappearing margin, the vertices on the farthest part of the margin to the joint part will be removed from the mesh. Whether these two parts are on the left or right part of the margin, it depends on the walking direction of the user. The algorithm 1 summarizes this procedure, which is shown in figure 8.

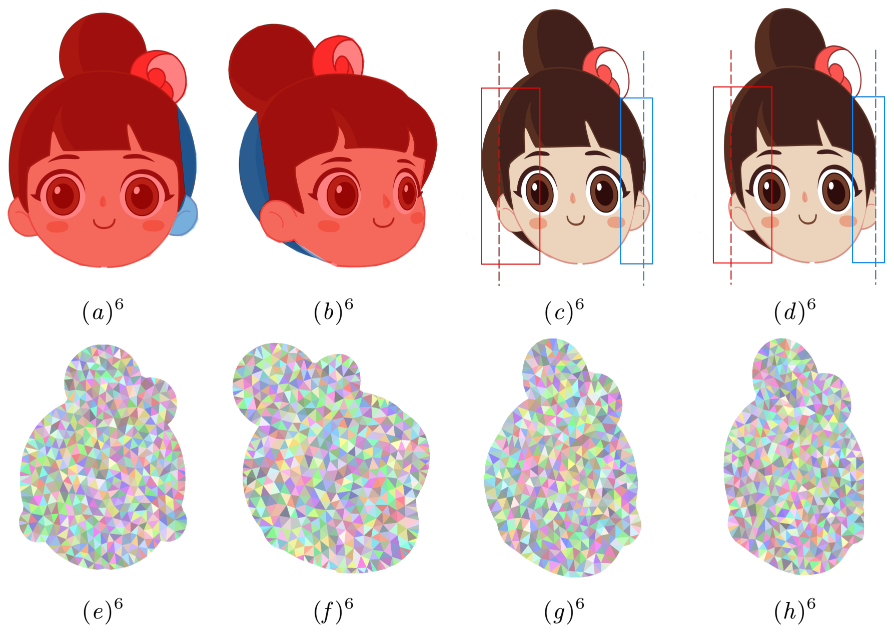


Fig. 9. An example of the interpolation process. In figures (a) and (b), the red sections are the joint parts, while the blue ones are the margins. Figure (c) shows the interpolated joint part merged with the two margins, while in figure (d) the margin are cut and it is shown the final interpolation at $t = 0.65$. It must be noticed that the deformation is on the mesh, not on the texture, that is just applied on it. In the second row, in the pictures (e), (f), (g), (h), it is shown the deformation process on the meshes.

The parallax scrolling works on a higher level. In fact, while the interpolation operates distinctly for each body part, and only if the user is between two thresholds, the parallax scrolling constantly moves all the body parts of the character. To each body part, as the user moves in the scene, is applied the translation in the equation 2.

As the character is a 2D figure, it lies on a 2D plane. The transformations applied by both the parallax scrolling and the shape interpolation do not add a third dimension to the mesh's vertices. Thus, the billboarding is just applied to the character's plane, and not individually to each component.

The whole procedure is performed every time the user moves, although the interpolation is executed only if the user is in particular positions, specifically between two thresholds. The algorithm 2 outlines the entire operation.

Algorithm 1 Body part interpolation

```
1: procedure InterpolateBodyPart(bodypart, angle)
2:     $t \leftarrow (angle - t_1)/(t_2 - t_1)$
3:     $bodypart_{jp} \leftarrow BDH(bodypart, t)$
4:     split margin at $x_{cut} \leftarrow m_{width \cdot t}$
5:     if margin is $am$ then                          ▷ in case of an *appearing margin*
6:         add to mesh vertices in closest part of the margin to $jp$
7:     else                                            ▷ in case of a *disappearing margin*
8:         remove from mesh vertices in farthest part of the margin to $jp$
9:     end if
10:    add vertices to mesh from the area $bodypart_{am} \cdot t$
11:    remove vertices from mesh from the area $bodypart_{dm} \cdot t$
12: end procedure
```

Algorithm 2 User's movement handler

```
1: procedure OnUserMovement
2:     $\vec{u}$ := vector between character's position and user's position
3:     $\vec{v}$ := character's viewing direction
4:     $angle \leftarrow \arccos(\vec{u} \cdot \vec{v})$
5:     for all $bodypart \in character$ do
6:         if $t_1 \leq angle \leq t_2$ then
7:             InterpolateBodyPart(bodypart, angle)
8:         end if
9:         $bodypart_{position} += (user_{translation} \cdot \lambda)/\zeta$
10:    end for
11:    rotate the character's plane around its central position towards the camera
12: end procedure
```

4 Results

In this section we discuss the experimental results we obtained while testing the proposed system with a 2D animation pipeline.

Figure 1 (a) and (b), as well figure 10, show how the different body parts move at the same time of the user, because of the parallax scrolling. The boy in figure 1 has the right arm and leg, as well as the eye and the mouth, in a front layer, hence the λ in the equation 2 is positive. The left arm and leg, instead, has a negative λ, as they are in the hindermost part of the character. In the end, the λ value for the torso and the head is 0. In the example in figure 1 the user is moving around the character in a counterclockwise sense – thus from the left to the right – therefore the body parts with positive λ are moving from the right to the left, in the opposite direction, while the ones in the rear side move in the same direction as the user. Moreover, as the forearm is closer to the user compared to the upper arm, it has a greater λ, therefore, it is moved more on the left.

Figure 9 shows instead an example of the interpolation of a body part between two different perspectives. In the figures (a), the source, and (b), the target, are shown the body parts from which the interpolation is computed. These two images represent the same body part, but from two different perspectives. In the two figures are also highlighted the joint part, in red, and the margins, in blue. In the figure (c), the shared part is interpolated using BDH shape interpolation, and the two margin are added to the body part. In the figure, the appearing margin is surrounded by the red box, while the disappearing margin by blue box. The dashed line marks the section of the margins that will be removed. In this particular example, $t = 0.65$. Figure (d) represents the final result of the computation. It can be noticed that there is some residue in the appearing margin, to the left of the dashed line. That is because that section of the hair is part of the joint section, thus it is not removed.

In the end, figure 10 a complete transaction of the camera from the front-right perspective to the front one. It can be noticed in figure (b), in particular on the torso and the head, the interpolation of the two states. It can be also noticed the parallax scrolling on the arms and the legs.

Our tool is implemented as a Unity3D native plug-in. This allowed us to use Nvidia's CUDA to parallelize on the GPU as many operations as possible, in particular the interpolation.

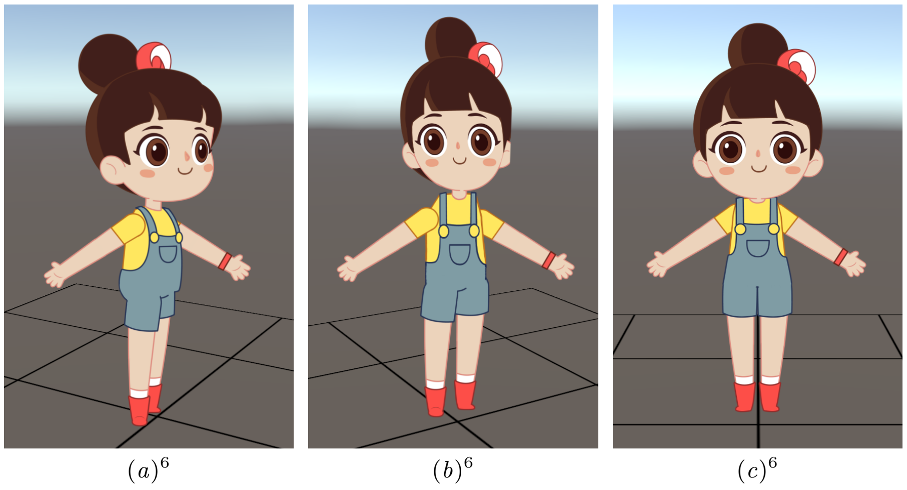

$(a)^6$ $(b)^6$ $(c)^6$

Fig. 10. A complete example of our system. Figure (a) shows the character from the front-right perspective, while in figure (b), it is in the middle of the transaction between that state and the front. Lastly, figure (c) shows the character from the front.

5 Conclusion

In this paper we presented an original tool for repurposing 2D characters and animations for a 3D VR environment. This method relies on three different technologies, the billboarding, the parallax scrolling and the 2D shape interpolation, which are combined together as illustrated by algorithm 2. This novel approach to content creation for VR represents a valid alternative to classic 3D characters, by offering a more economic and faster way to produce products for the platform.

Although the system produces good results, we are aware of few limitations that affects it. For instance, at the moment the identification of the joint part and the margins of each body part must be manually set by the user. Moreover, as the system is focused on the repurposing of 2D characters, the other objects in the scene are 3D. There are two ways to handle the interaction of the 2D character with the scenery. The first solution is to extend the system to make it work with every kind of objects; the second one is to use the 3D surroundings and to study a way to make them interact with the 2D character in a plausible way. Ultimately, it would be interesting to make a comparison with other 2D mesh interpolation methods.

References

1. Akenine-Moller, T., Haines, E.: Real-Time Rendering. A. K. Peters, Ltd., 2nd edn. (2002)
2. Alexa, M.: Recent advances in mesh morphing. In: Computer graphics forum. vol. 21, pp. 173–198. Wiley Online Library (2002)
3. Alexa, M., Cohen-Or, D., Levin, D.: As-rigid-as-possible shape interpolation. In: Proceedings of the 27th annual conference on Computer graphics and interactive techniques. pp. 157–164. ACM Press/Addison-Wesley Publishing Co. (2000)
4. Asbury, K., Cook, L., Howard, M., Katzenber, J., Soria, M.: Spirit: Stallion of the cimarron. DreamWorks Pictures (2002)
5. Aubel, A., Boulic, R., Thalmann, D.: Real-time display of virtual humans: levels of details and impostors. IEEE Transactions on Circuits and Systems for Video Technology 10(2), 207–217 (2000)
6. Balkan, A., Dura, J., Eden, A., Monnone, B., Palmer, J.D., Tarbell, J., Yard, T.: Parallax scrolling. In: Flash 3D Cheats Most Wanted, pp. 121–164. Springer (2003)
7. Barbieri, S., Garau, N., Hu, W., Xiao, Z., Yang, X.: Enhancing character posing by a sketch-based interaction. In: ACM SIGGRAPH 2016 Posters. p. 56. ACM (2016)
8. Baxter, W., Barla, P., Anjyo, K.i.: Rigid shape interpolation using normal equations. In: Proceedings of the 6th international symposium on Non-photorealistic animation and rendering. pp. 59–64. ACM (2008)
9. Bird, B., Abbate, A., McAnuff, D.: The iron giant. Warner Bros. Feature Animation (1999)
10. Blow, J.: Braid (2008), `http://braid-game.com/`
11. Chen, R., Weber, O., Keren, D., Ben-Chen, M.: Planar shape interpolation with bounded distortion. ACM Transactions on Graphics (TOG) 32(4), 108 (2013)
12. Chien, E., Chen, R., Weber, O.: Bounded distortion harmonic shape interpolation. ACM Transactions on Graphics (TOG) 35(4), 105 (2016)
13. Choi, J., Szymczak, A.: On coherent rotation angles for as-rigid-as-possible shape interpolation. Tech. rep., Georgia Institute of Technology (2003)
14. Clements, R., Musker, J., Conli, R.: The treasure planet. Buena Vista Pictures (2002)

15. Cooper, D.: 2d/3d hybrid character animation on spirit. In: ACM SIGGRAPH 2002 conference abstracts and applications. pp. 133–133. ACM (2002)
16. Davis, J., Agrawala, M., Chuang, E., Popović, Z., Salesin, D.: A sketching interface for articulated figure animation. In: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation. pp. 320–328. Eurographics Association (2003)
17. Fox, T.: Undertale (2015), `http://undertale.com/`
18. Guay, M., Cani, M.P., Ronfard, R.: The line of action: an intuitive interface for expressive character posing. ACM Transactions on Graphics (TOG) 32(6), 205 (2013)
19. Hahn, F., Mutzel, F., Coros, S., Thomaszewski, B., Nitti, M., Gross, M., Sumner, R.W.: Sketch abstractions for character posing. In: Proceedings of the 14th ACM SIGGRAPH/Eurographics Symposium on Computer Animation. pp. 185–191. ACM (2015)
20. Jain, E., Sheikh, Y., Hodgins, J.: Leveraging the talent of hand animators to create three-dimensional animation. In: Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation. pp. 93–102. ACM (2009)
21. Jain, E., Sheikh, Y., Mahler, M., Hodgins, J.: Augmenting hand animation with three-dimensional secondary motion. In: Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation. pp. 93–102. Eurographics Association (2010)
22. Konno, H., Miyamoto, S.: Mario kart 64. Nintendo (1996)
23. Lipkin, N.: Examining indie's independence: The meaning of" indie" games, the politics of production, and mainstream cooptation. Loading... 7(11) (2012)
24. Mao, C., Qin, S., Wright, D.: A sketch-based gesture interface for rough 3d stick figure animation. Eurographics (2005)
25. O'Hailey, T.: Hybrid animation: integrating 2D and 3D assets. Taylor & Francis (2010)
26. Poranne, R., Lipman, Y.: Provably good planar mappings. ACM Transactions on Graphics (TOG) 33(4), 76 (2014)
27. Schaufler, G.: Dynamically generated impostors. In: GI Workshop Modeling-Virtual Worlds-Distributed Graphics. pp. 129–136 (1995)
28. Scherba, T.: Virtual reality is about to go mainstream, but a lack of content threatens to hold it back. `https://techcrunch.com/2016/04/03/virtual-reality-is-about-to-go-mainstream-but-a-lack-of-content-threatens-to-hold-it-back/` (2016)
29. Sỳkora, D., Sedlacek, D., Jinchao, S., Dingliana, J., Collins, S.: Adding depth to cartoons using sparse depth (in) equalities. In: Computer Graphics Forum. vol. 29, pp. 615–623. Wiley Online Library (2010)
30. Tecchia, F., Chrysanthou, Y.: Real-time rendering of densely populated urban environments. In: Rendering Techniques 2000, pp. 83–88. Springer (2000)
31. Weber, O., Ben-Chen, M., Gotsman, C.: Complex barycentric coordinates with applications to planar shape deformation. In: Computer Graphics Forum. vol. 28, pp. 587–597. Wiley Online Library (2009)
32. Weber, O., Gotsman, C.: Controllable conformal maps for shape deformation and interpolation. In: ACM Transactions on Graphics (TOG). vol. 29, p. 78. ACM (2010)
33. Xu, D., Zhang, H., Wang, Q., Bao, H.: Poisson shape interpolation. Graphical models 68(3), 268–281 (2006)

34. Yang, Y., Wang, X., Chen, J.X.: Rendering avatars in virtual reality: integrating a 3d model with 2d images. Computing in Science & Engineering 4(1), 86–91 (2002)
35. Zemeckis, R., Marshall, F., Watts, R.: Who framed roger rabbit. Buena Vista Pictures (1988)