Facial pipeline in Playmobil: the movie

Jeremy Ringard
ON Animation studios

Claude Levastre
ON Animation studios

ABSTRACT

In this paper, we present the technical pipeline that has been deployed at ON Animation studios to manage the specificity of facial animation on the Playmobil movie. According to the artistic requirement of this production, we developed a complete texture-free solution that gives the artists the ability to animate 2D facial features on a three-dimensional face while having a realtime raytraced feedback in the viewport. This approach provides full control over the shapes and since the final result is computed at render time, the visual style can be controlled until the very end of the workflow.

CCS CONCEPTS

• **Computing methodologies** → **Animation**; **Rendering**.

KEYWORDS

ray tracing, animation, facial, 2D, projection, texture

ACM Reference Format:
Jeremy Ringard and Claude Levastre. 2019. Facial pipeline in Playmobil: the movie. In *Proceedings of SIGGRAPH '19 Talks.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3306307.3328207

SIGGRAPH '19 Talks, July 28 - August 01, 2019, Los Angeles, CA, USA

ACM ISBN 978-1-4503-6317-4/19/07.
https://doi.org/10.1145/3306307.3328207

1 INTRODUCTION

Facial animation for CG characters has been a well discussed topic in the past years. Many approaches have been developed to make the deformation of 3D facial features user friendly. The production of the movie Playmobil raises a new challenge: According to the toy's design, the facial features are flat, made of paint applied directly to the face geometry. Thus, we had to design a new approach to fulfill this requirement. We also had to make sure the artistic choice wouldn't be constrained by technical issues. That means we want our system to be robust enough to display complex face setups on a possibly deformable head shape, and flexible enough not to force the animation style into stepped motion or too exotic rigging system.

2 CANDIDATE SOLUTIONS

Several solutions have been considered during the design process of our facial setup. Here is a quick overview of the reasons why we eliminated them.

2.1 Projected geometry

Modeling and rigging facial features as simple shapes projected on the face's surface would provide a quick visual feedback to the animators but this solution suffers Z-fighting artifacts. Projecting 3D shapes on the face could also have intersection or floating issues on close up shots. Moreover, this approach is more suited for characters with very simple facial features such as a black dot for the eye. Building any complex composition is much more complicated: For instance, having an eyelid that masks the eye while the pupil's visibility is clamped by the limits of the visible sclera would require a much more complicated rig.

2.2 Texture flipbook

Another natural approach would consist in using a catalog of textures for the different facial expressions, possibly split per facial feature. This "flipbook" like solution makes the rigging step much easier, but it comes with several drawbacks. Although it helps the animators to keep their animation coherent with the overall look of the movie, it also limits the amount of expressions available and doesn't provide any flexibility to add subtle variations along the animation. This solution could also become costly regarding storage, texture resolution and overall file management.

3 OUR SOLUTION: 3D RIGGED FLAT SHAPES

The solution we developed is designed in three parts: First, a 3D rig driving 2D shapes in the animation software. The second part is a custom Maya plugin that converts these shapes to a dynamic texture and apply it on the 3D face for viewport display. The final part is a specific shader executed at render time in Guerilla render in order to composite the facial features without resolution or sampling limitations.

3.1 Facial rig

Given the specific 2D style of the facial, our workflow doesn't need to rely on a modeling step for the face. Contrary to most common CG characters, the setup of the face can go straight from design to the rigging team. Thus, we developed a set of tools inside Maya that give the rigging artists the ability to draw directly the shapes on the face, matching the provided reference. Since the complexity of the face relies mostly on the layering of the shapes rather than on the shapes themselves, the artists just have to draw curves matching the silhouette of each facial feature.

These manually shaped curves are then converted into flat Bezier curves in a 2D space corresponding to the head UV space, and each facial feature is sorted on the Z axis to provide basic layering (ex: pupil in front of the sclera). This curve system is automatically rigged with generic parenting relationships, and any specific parenting or layering requirement could be scripted directly in the rig asset. this last feature allows us to rebuild the rig on the fly without further tweaking. This full Bezier approach is interesting because it allows the animator to control every control point and tangent of the curves, giving them the ability to create virtually any shape. Moreover, the 2D version of this rig is totally transparent to the user: The controllers of the rig are located on the head of the 3D character and the animator doesn't need any separated interface for the face. From the user's point of view, the facial rig works just like any 3D rig.

3.2 realtime facial feedback

The next step is to provide the animators a real-time feedback of the character's face in order to have a WYSIWYG (What You See Is What You Get) solution. Basically, the Bezier curves manipulated by the animators are lofted to generate flat geometry in UV space and the resulted shapes are plugged in a custom Maya node: This "render and compose to texture" node takes the 2D shapes as inputs, and generate a texture of the final face stored in VRAM. This texture is plugged directly in the shader of the character to display an accurate representation of the face in maya's viewport. This compositing is executed in realtime, at a resolution that could be changed anytime by the animators according to their own needs. Although the shapes are ordered along the Z axis, the system cannot rely just on this to composite the face. For instance, the pupil must be clipped when it's outside of the sclera (fig.1). In order to solve this issue, the compositing node also uses binary masks associated with each shape. The internal algorithm uses this mask to define the visibility of the shapes for each pixel of the texture. The node performs a grid accelerated raycasting (computed on the GPU) that goes through every geometry and compose the final pixel color in a single pass. Using these mask values on each facial feature of the rig allows us to quickly provide an accurate display of the final face.

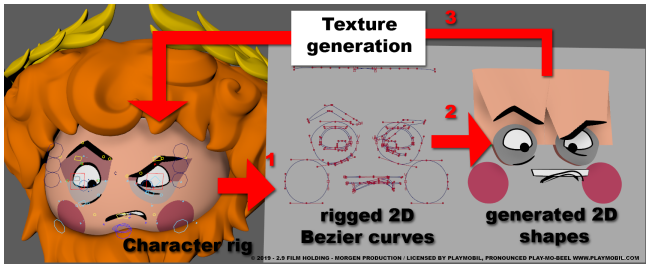

Figure 1: the viewport generation process

3.3 final render

The final part of the solution is focused on maximizing the flexibility of the process until the render time of the shot: Instead of storing and transferring the animated textures from Maya to the rendering engine, we transfer the animated 2D shapes directly with the rest of the character into Guerilla Render. The shapes are then processed by a custom shader performing basically the same process as the Maya node, except that instead of feeding a virtual texture, the shader applies on a traceset. This process can be seen as follows: during rendering, each time a ray hits the surface of the face, the facial traceset is queried and a new ray is shot along the Z axis to the 2D shapes.

Using such a process has multiple advantages: first, it's 100% texture free since the facial animation is transferred from animation in the same alembic file as the body. Second, this approach gets rid of any question regarding texture resolution: since the traceset is computed for every pixel of the face, the resolution is virtually unlimited. Also, since we are using actual geometry for this sampling, effects such as motion blur can be activated and controlled directly at rendertime.

4 CONCLUSIONS AND FUTURE WORK

This process has been successfully used during all the production of Playmobil. Among the improvements we are planning for the future, we want to unify the code by merging the rasterizer node and the render shader into a single library.

ACKNOWLEDGMENTS

The authors would like to thank Pascal Bertrand, Rachid Chikh and Olivier Rakoto.