PRIM project: Pixel Art Manipulation

Supervisor: Amal Dev Parakkat
By student: Vannvatthana Norng
Télécom Paris

28 mars 2024

1 Introduction

The 8-bit era was when the famous titles such as Super Mario Bros., Legend of Zelda, Dragon Quest, etc made their first appearences, and it laid some groundwork in the gaming industry for fantastic video games that we enjoy today. Nowadays, with the advancement in graphics and hardwares, the visual in video games looks more and more smooth, realistic, and has high resolutions. However, 8-bit style graphics still attracts artists, developpers and consumers, including myself with its unique retro style, and therefore, pixel art still holds a place in the art and gaming community.

Same as other forms of 2D animations, an artist needs to draw many frames of a pixel art character with different poses and put those frames together in order to make one second of the charactor moving, looking from one view. In this project, I attempted to propose a solution where the user only needs to draw a frame of a character with a sample pose, then manipulate that character into different poses to make all the frames needed for animation. The solution will follow the steps below :

1. The user draws the character with T-pose from different views
2. Those views are used to build a 3D voxel model of the character
3. The 3D voxel can be deformed and changed into different poses
4. A 2D pixel art frame is then exported for any view desired

2 Constructing 3D Voxel from 2D pixel art images

The goal of constructing 3D voxel model of the character instead of just moving the pixel informations in 2D is to ensure that the interpolation of the pose between views are coherent to each other. In order to construct 3D voxel of a character, I used four different views of the caracter in T-pose : front, back, top, and side views (figure 1) with the same square image resolutions. The construction starts by using the front and back views to build the model's shape along the Y-axis (figure 2), and then the side and top views are projected to clean off the excess voxel cubes that are not needed in order to shape the voxel as it should be. It is possible to not use the back view for constructing the 3D model, but it is here for the detail purpose. After successfully, the voxel model should look like in figure 3.

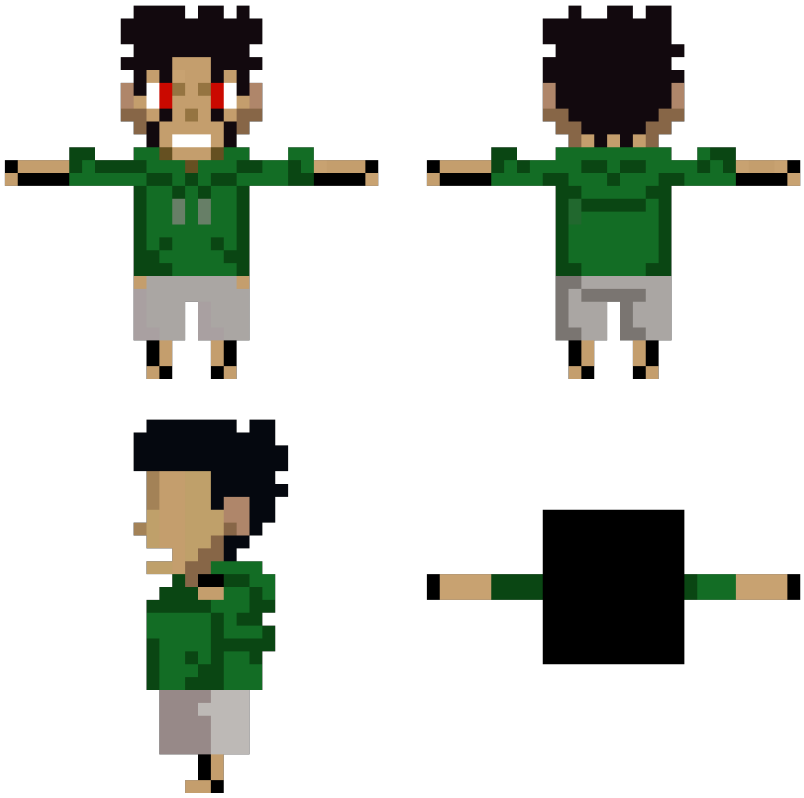

FIGURE 1 – Front, Back, Side and Top view of a character for building 3D Voxel

How the voxel should be manipulated

The goal of the rules that I considered for my voxel models was so that they can be manipulated correctly in 3D spaces while respecting the constraints of 2D pixel arts, and therefore after the character is in a desired pose, the projection results back to 2D pixel images do not need too much inpainting.

Deforming a voxel model with a constraint of 2D images is different from deforming a normal 3D mesh. When this type of voxel model is deformed, every voxel cube needs to stay in positions where (x, y, z) coordinates are all integers as the cubes represent as the RGB color information for a whole pixel (u, v) from one view. As a result, each cube has only one color, and so it contains two important information : the integer coordiates of the center of the cube, and its RGB value. The manipulation of the model is the same as moving only the center of each cube around independently, relative to the world space, which is why the coordinates of its vertices are not needed for this procedure. In pixel art, when two colored pixels collide together(e.g. an arm is folded, some pixels around the elbow collide), one of them just hide behind the other one. To consider that condition in 3D, in case there are two voxel cubes collides in the same position, they can just stay there together without any problems.

3 Solutions and implementation

For this implementation, I used the CGP library [2] from Mr. Damien Rohmer as it already has the necessary tool to render each cube independently with the information of

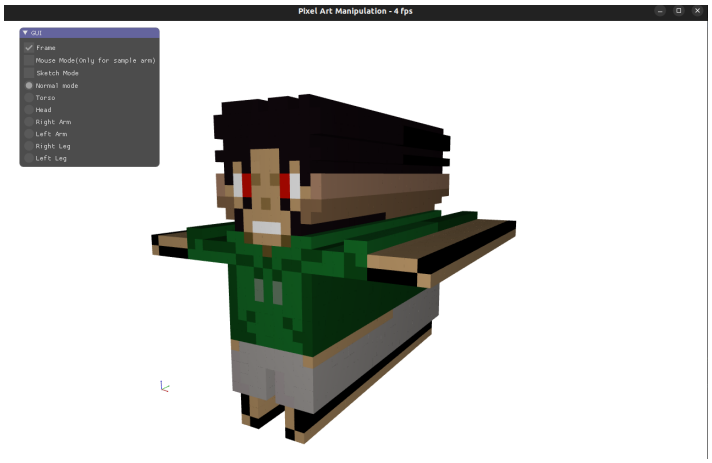

FIGURE 2 – Before cleaning with top and side view

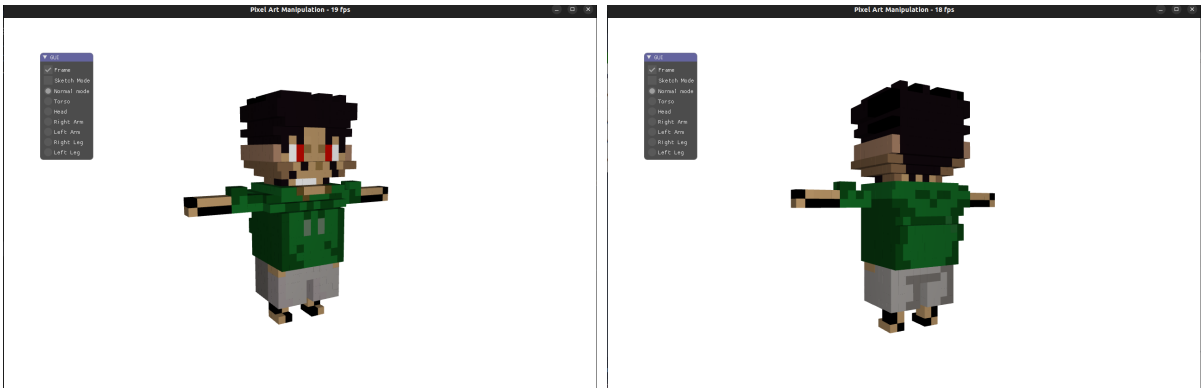

FIGURE 3 – Constructed 3D Voxel

its center's position, size, and RGB color. Each cube is a unit cube (edge length is 1) in the world space. I came up with two solutions for manipulating this kind of voxel.

3.1 Spring Voxel

The first implementation that was attempted for manipulation of the constructed voxel model is by using spring simulation. Every pair of neighbor cubes have their centers connected with a spring(figure 4), then I applied Hook's Law for each spring where the restoring force follows this formula :

$$F = -kx \tag{1}$$

where k is spring constant, the spring is harder to compress or decompress when k is larger. x is displacement, it indicates how far the spring is compressed or decompressed compare to its rest length, and the minus sign indicates that the force is always in the opposite direction of displacement.

Finally, for each voxel cube, we do the sum of all forces that are applied on the center (figure 4)

$$\vec{F_i} = \vec{F_{i1}} + \vec{F_{i2}} + \vec{F_{i3}} + \vec{F_{i4}} + \vec{F_{iex}} \tag{2}$$

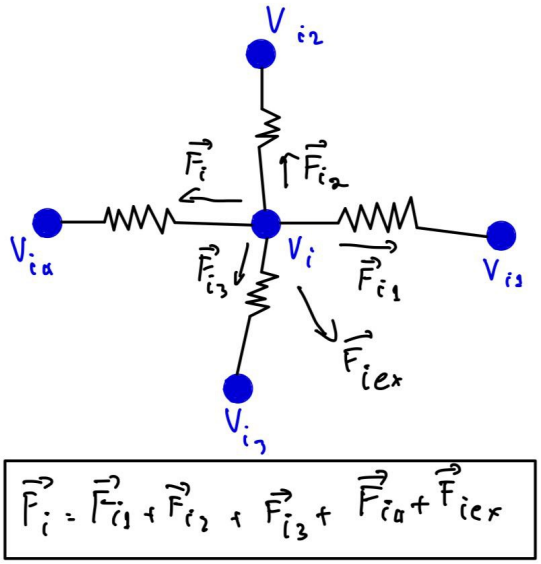

FIGURE 4 – All forces that affects the cube V_i ($\vec{F_{iex}}$ is the exterior force applied to the spring system

The computation of the spring force for the entire body is quite expensive, however it could be manipulated properly without any problem. Desprite the easiness of the implementation, the constant rest length for each spring keeps the voxel cubes that are being manipulated from staying in the integer coordinates due to different distances between neighbors. Let us look at the centered cube in figure 5. At resting position, the distance between 2 centers of cubes that share a face is 1 unit in world space, for 2 cubes that only share an edge has a distance of $\sqrt{2}$ unit, and 2 cubes that only share a vertex has a distance of $\sqrt{3}$ unit between their centers, so the rest lengths for them are 1, $\sqrt{2}, \sqrt{3}$ respectively.

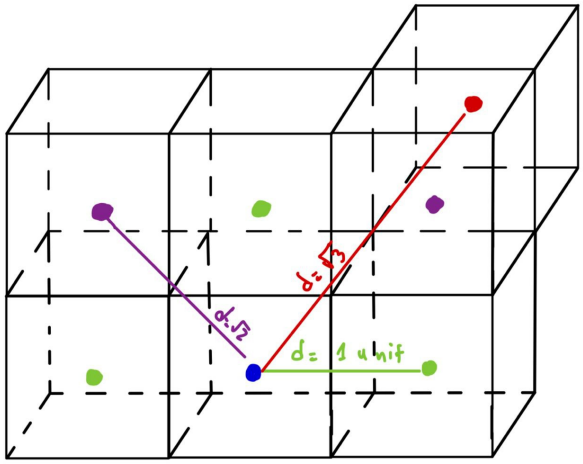

FIGURE 5 – Distance from the center of 1 cube to its neighbor at initial state

To be able to make another posture for the character, these cubes need to adapt to the integer coordinates so that the model's visual stays within pixel art's constraints. With spring simulation, the voxel cubes try to stay at the same distances from all of their neighbors, which is why it creates holes at the body parts that the model is rigged (figure6.

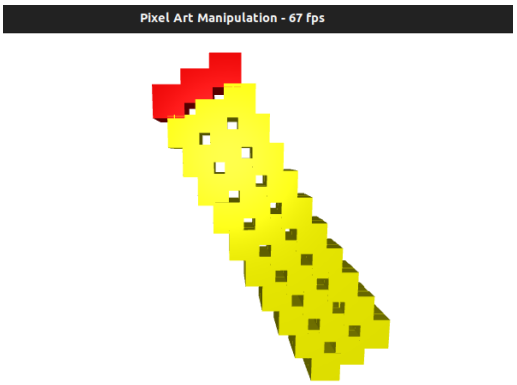

FIGURE 6 – Spring system causes the cube to not stay in integer positions and creates holes

Because of the computational cost and the struggle of voxel cubes to always stay in integer coordinates, the spring simulation is not suitable for manipulating pixel art in 3D space.

3.2 Skeleton Voxel

The second approach which I attempted to manipulate the voxel model was by using skeleton. For each bone of the skeleton, the informations needed for each bone are positions of the start joint and the end joint, they are used for computing the 4D global transformation matrix of the local space of that bone. Firstly, the direction from start joint to end joint is chosen for Z axis :

$$\vec{u_z} = \frac{\text{endjoint} - \text{startjoint}}{|(\text{endjoint} - \text{startjoint})|} \tag{3}$$

. In order to compute the two other unit vectors $\vec{u_x}, \vec{u_y}$, I chose a constant vector $\vec{I} = (1, 1, 1)$, then calculate the cross products as follow :

$$\vec{u_x} = \frac{\vec{I}}{|\vec{I}|} \times \vec{u_z} \tag{4}$$

$$\vec{u_y} = \vec{u_z} \times \vec{u_x} \tag{5}$$

The local transformation space of the bone is $\vec{u_x}, \vec{u_y}, \vec{u_z}$ as its x, y, z axis with start joint as the local origin. The global transformation matrix is then calculated by the translation and rotation from the world space to the local space. Finally, a set of voxel cubes are assigned to be associated with the bone so that they could move along with the bone in

world space. Their local positions with respect to the local space of the bone are calculated as follow

$$(x_{\text{local}}, y_{\text{local}}, z_{\text{local}})^T = (\text{Global Transformation Matrix})^{-1} \times (x_{\text{global}}, y_{\text{global}}, z_{\text{global}})^T \quad (6)$$

When the bone is manipulated, we just update the new global transformation matrix, then update the new global positions of associated cubes using local coordinates calculated earlier with the reverse of equation (6) :

$$(x_{\text{global}}, y_{\text{global}}, z_{\text{global}})^T = (\text{Global Transformation Matrix}) \times (x_{\text{local}}, y_{\text{local}}, z_{\text{local}})^T \quad (7)$$

Each coordinate x_{global}, y_{global}, z_{global} is then rounded to the nearest integer.

Adding sketch control

I found the mouse's position projection from screen space to world space is quite hard to control for my manipulation, therefore I implemented the sketch option to manipulate more easily the limbs of the character to any directions that I want. It is inspired by [3], but instead of taking several points on the sketch, I implemented a more simple one with taking only the start and end points of the sketch as a guide. The vector $\vec{\text{dir}} = \text{end point} - \text{start point}$ is used as the new vector $\vec{u_z}$ of the bone that is being manipulated, indicating the new direction of the bone. Then, we follow the steps from equations (3, 4, 5, 7) again to update the positions of associated cubes.

Results

While it gave better results than the spring voxel, the manipulated bone only looks fine in some specific positions such as ± 90 and ± 180 degrees compare to each axis of the world space (figure 7). However, the voxel starts breaking and looking different from its original form in other angles as in figure 8. This is because of the rounding of coordinates to the nearest integers, and I have not found a right way to make it work yet.

FIGURE 7 – Case where the manipulation works

FIGURE 8 – The arms' forms are changed

4 Conclusion

In conclusion, I have attempted to manipulate pixel art characters by projecting them into 3D voxel meshes, and then deforming them while respecting the constraints of 2D pixel images. The two solutions that I tried were unfortunately not working, but I am still positive that it is going in the direction where it should be. For the future improvement of this project, I've considered continuing with the second solution of skeleton voxel with more input parameters that are necessary to help guide the voxel cubes to better positioning for each limb of the voxel mesh. However, I am also concerned that it might goes far from the original purpose to make the voxel manipulable right from the start.

References

1. Sample pixel art character used in this project is from a CTN-Artist's post in Reddit.
2. CGP Library by Damien Rohmer : https://github.com/drohmer/cgp_examples
3. Martin Guay, Marie-Paule Cani, Rémi Ronfard (2013)- The line of action: an Intuitive Interface for Expressive Character Posing