The Tech and Art of Cyberspaces in Cyberpunk 2077

Oskar Świerad
oskar.swierad@cdprojektred.com
CD PROJEKT RED
Warsaw, Poland

Peter Ankermann
peter.ankermann@cdprojektred.com
CD PROJEKT RED
Warsaw, Poland

Krzysztof Krzyścin
krzysztof.krzyscin@cdprojektred.com
CD PROJEKT RED
Warsaw, Poland

Figure 1: Johnny Silverhand in cyberspace.

ABSTRACT

A deep dive into the technology and art behind cyberspace and braindances in *Cyberpunk 2077*. Braindances are the recorded memories and feelings of individuals, reprojected in the mind of the viewer. To bring this concept into reality, we decided to follow an unconventional approach to rendering environments and characters in real-time. The core visual concept was based around sparse point clouds and glitch effects. Post processes like datamoshing were used to further hide the underlying geometry, aiming for a surreal, out-of-body experience.

CCS CONCEPTS

• **Computing methodologies → Computer graphics**.

KEYWORDS

games, realtime rendering, visual effects, shader, post process, point cloud

ACM Reference Format:
Oskar Świerad, Peter Ankermann, and Krzysztof Krzyścin. 2021. The Tech and Art of Cyberspaces in Cyberpunk 2077. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Talks (SIGGRAPH '21 Talks), August 09-13, 2021.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3450623.3464662

1 INTRODUCTION

In this talk, we explain how cyberspace and braindance features were developed from a technical artist's point of view. Sparse 3D point clouds were used to visualize the virtual environments of both types of digital spaces. Real-time photogrammetry was used in braindance sequences to reconstruct the position and color of the scene in the form of a point cloud. As for post processes in cyberspace, we applied pixel sorting and datamoshing to reach our vision of the virtual worlds in *Cyberpunk 2077*.

2 ENVIRONMENT AND BRAINDANCES

We used point cloud rendering as the core visual idea behind cyberspace and braindances. Scenes are captured by rasterization from multiple viewpoints in a process similar to photogrammetry. The resulting points positions and colors are stored as textures, to be interpreted by shaders in the game.

Figure 2: Braindance tutorial scene, as seen in the game.

Cyberspace and braindance implementations differ in several areas. The most important one is the distinction between static scene capturing (cyberspace), performed locally on the developer's workstation, and a dynamic one, happening in real-time in the game (braindance). The static approach allows us to pre-process the data in as time-consuming a process as needed, as the cost of the operation doesn't affect the final real-time performance. The output is stored as textures where 1 pixel represents 1 point (quad) of the cloud.

The input points were generated in external photogrammetry software, from as many angles as needed, based on videos captured in the game engine. SideFX Houdini was then used for post-processing the results. Our Houdini batch process removed points with a low contribution to the final image through heuristics like luminosity, closeness to other points, and occlusion culling. We also implemented 3D spatial clustering in Houdini. It splits a single big

---

SIGGRAPH '21 Talks, August 09-13, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8373-8/21/08.
https://doi.org/10.1145/3450623.3464662

point cloud, representing the entire scene, into smaller, spatially coherent chunks. This allows the game to cull the visibility of the static point clouds based on their bounding boxes.

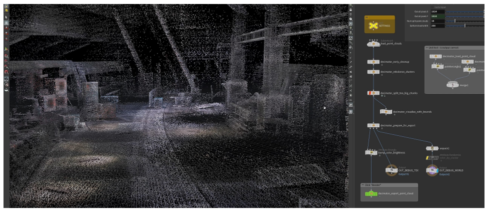

Figure 3: Editing the point cloud in SideFX Houdini.

The dynamic method of braindance captures points positions and their color in real-time. This allows vehicles, people, and objects to move. Changes in lighting conditions are also reflected instantly in the visible point cloud. We achieved that by placing 2 or 3 cameras in every braindance location in the game. These virtual cameras render the same scene as the main one, but have lower resolution and narrower clipping distances, for a smaller performance footprint.

Only the Z depth of the scene is captured, to be translated into 32-bit floating point world-space position and stored as a texture. The resulting image represents the camera's continuous view space. Each camera gets its own instanced mesh of quads. Such a point cloud mesh has procedurally generated UV coordinates that match the view area of the camera.

The color of the dynamic point cloud is calculated in the pixel shader. Then it is used to fetch the final scene color from the game's buffers, after lighting but before post processes. This color is processed by applying 3D noises and a precomputed shadow mask to get a desired artistic effect, then applied to the quad.

3 CHARACTER SHADER

The holographic representation of the characters in cyberspace was done with a special shader for these characters. An opaque blending mode with a 1-bit alpha mask is used. This kind of opacity is fast to render and avoids the sorting issues that are common with alpha-blending translucency modes in real-time engines. To achieve a soft opacity mask despite the binary alpha, Bayer dithering was applied to it, in combination with temporal anti-aliasing [Korein and Badler 1983].

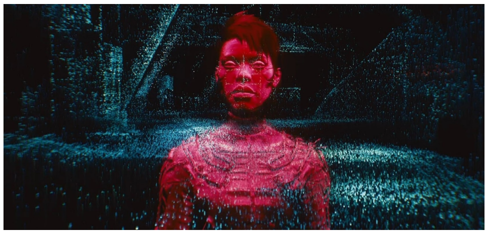

Figure 4: Brigitte with applied cyberspace character shader.

In addition to this technique, a mask based on the Fresnel term was applied to reduce the visibility of the edge of the surface parallel to the camera. For readability purposes, an additional low-resolution texture mask for the head was introduced to limit the undesired effect of the Fresnel mask hiding parts of the face. An animated 3D noise map, stored as a tileable 2D texture array, was used to randomize the transparency effect even more.

4 POST PROCESSES

A multitude of standard post-processing effects were used for cyberspace. Examples include bloom, tinting, chromatic aberration, and color grading, to name a few. Then we developed real-time implementations of datamoshing and pixel sorting. These two methods are common in offline video post-processing but are rarely found in video games due to the computing power and memory requirements of their usual implementations.

Our pixel sorting shader does what the name already suggests, sorting the pixels of the screen color buffer - in our case vertically - by luminosity. Use of GPU-friendly sorting algorithms in combination with a compute shader helped maintain a real-time frame rate.

Datamoshing refers to video compression errors caused by unreadable I-frames. Video compression works with 3 types of key frames: I, P, and B-frames. I-frames contain complete images, but are only stored sparsely and when the picture changes abruptly. P and B-frames, on the other hand, are used in between the I-frames and only store instructions to get from one complete image to the next. If an I-frame can't be read anymore because of corruption, it will appear as if the image is projected onto another background, or it starts to smear, until the next I-frame can be parsed correctly.

Figure 5: Feedback loop effect generated by datamoshing.

To deliberately replicate this effect, we applied multi-pass rendering to store the last frame of the rendered color buffer and blended it with the current one via a noise mask. In addition to this, the motion vectors were read and applied to transform the image of the last frame before these two images were blended together. This post-process went through multiple iterations and the softer dream-like transition ended up in the final game.

5 CONCLUSION

We discussed the artistic decisions and technical solutions that we ended up with regarding cyberspace and braindance visualization. After a lot of iteration on every aspect, in close cooperation with our rendering department, we were able to realize a never-before-seen representation of digital realities in a 3D video game.

REFERENCES

Jonathan Korein and Norman Badler. 1983. *Temporal anti-aliasing in computer generated animation.* University of Pennsylvania. https://dl.acm.org/doi/10.1145/964967.801168