CR-PLAY

"Capture-Reconstruct-Play: an innovative mixed pipeline for videogames development"

Grant Agreement
ICT-611089-CR-PLAY

Start Date	End Date
01/11/2013	31/10/2016

Deliverable 4.4

High-fidelity Prototypes of mixed pipeline for videogame development

Document Information

Deliverable number:	4.4
Deliverable title:	High-fidelity Prototypes of mixed pipeline for videogame development
Deliverable due date:	31/08/2015
Actual date of delivery:	31/08/2015
Main author(s):	Ivan Orvieto, Matteo Romagnoli (TL)
Main contributor(s):	George Drettakis, Fabian Langguth, Michael Goesele (TUD), Corneliu Ilisescu, Gabriel Brostow (UCL)
Version:	1.0

Versions Information

Version	Date	Description of changes
0.1	22/07/2015	Structure and contributors
0.2	13/08/2015	Contributions integrated
0.3	21/08/2015	Pre-final version for internal review
1.0	31/08/2015	Final version

Dissemination Level

PU	Public	**X**
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Deliverable Nature

R	Report	
P	Prototype	**X**
D	Demonstrator	
O	Other	

CR-PLAY Project Information

The CR-PLAY project is funded by the European Commission, Directorate General for Communications Networks, Content and Technology, under the FP7-ICT programme.
The CR-PLAY Consortium consists of:

Participant Number	Participant Organisation Name	Participant Short Name	Country
	Coordinator		
1	Testaluna S.R.L.	TL	Italy
	Other Beneficiaries		
2	Institut National de Recherche en Informatique et en Automatique	INRIA	France
3	University College London	UCL	UK
4	Technische Universitaet Darmstadt	TUD	Germany
5	Miniclip UK Limited	MC	UK
6	University of Patras	UPAT	Greece
7	Cursor Oy	CUR	Finland

Summary

This is the fourth deliverable of **Work Package 4: "Design and Development (Requirements, Functional Specifications, and Prototypes)"**. The leader of this work package is **TL**, with involvement of all other partners. The objective of this work package is focused on gathering end-user needs, forming these into functional specifications, and creating the prototypes of the mixed pipeline for videogame development. This WP sets into practice the *user-centred design* approach adopted by CR-PLAY, ensuring that the technologies developed will result in tools that are effective and usable for professional and semi-professional use.

This deliverable - **"D4.4 High-fidelity Prototypes of mixed pipeline for videogame development"** -describes the results of **Task4.4 "High-fidelity Prototypes"**. The document presented here is meant to play along with the High-fidelity software package explaining the main features developed starting from the Low-Fidelity prototype (D4.3), their integration and communication in the common platform, risks and related contingency actions.

The structure of this deliverable is as follows:

Section 1 describes the technological upgrade provided by Unity 5 and how it influenced the development of this phase of the CR-PLAY mixed pipeline.

Section 2, 3 and 4 present the integration of the features from the Capture and Reconstruct (WP1), Image Based Rendering (WP2) and Video Based Rendering (WP3) pipeline steps.

Section 5 presents the main risks and related contingency actions that are foreseeable at this stage of the project. They will be constantly updated as long as development activities progress during the project span.

Finally, Section 6 draws conclusions and describes next steps of development activities in WP4.

Table of Contents

Abbreviations and Acronyms

- **GLSL:** OpenGL Shader Language
- **GPU**: Graphics Processing Unit
- **IBR:** Image Based Rendering
- **IL:** Intermediate Language
- **PMVS:** Patch-based Multi-View Software
- **RLE**: Run Length Encoding
- **SDK:** Software Development Kit
- **SfM;** Structure from Motion
- **SP:** SuperPixels
- **VBR:** Video Based Rendering
- **WP:** Work Package

Introduction

Starting from results of Task 4.3 (that led to the release of the Low-fidelity prototype at M12), in Task 4.4 we continued the work on integration of software modules created in WP1, WP2 and WP3 and on development of new and improved features of the pipeline, always paying attention to quality, speed and optimizations, and taking user feedback into account.

Based on the results of Low-fidelity prototype evaluation (MS13), an analysis and prioritization of the feedback coming from game developers has been done, with the consequent definition of the roadmap towards the creation of the High-fidelity prototype (MS11). **The main goal was the delivery of an advanced version of the pipeline based on Capture and Reconstruction, IBR and VBR, that allows game developers to create game prototypes during the subsequent evaluation activities.**

A constant monitoring of trending platforms (mobile in particular), technologies and products in the videogame market, has helped the partners involved in WP4 to more precisely define how to prioritise features and their development (in accordance with project's requirements), towards **a concrete release of the pipeline that could appeal the community of game developers.**

In parallel, a thorough technical analysis (Task 4.6) highlighted the weaknesses and bottlenecks of the Low-fidelity prototype of the pipeline, contributing to the definition of the (several) refactoring interventions focused on reduction of used data and rationalisation of stored information thus leading to **a significant reduction of memory usage and disk space occupation.**

Additionally, a detailed investigation pointed out the main causes of artifacts and quality loss. This allowed the definition of lines of intervention in order to mitigate the disturbing effects and thus improving the final quality of rendering, so as **to provide a low-cost photorealistic solution for game development**.

Finally, new techniques have been developed to **improve the depth interaction between Image Based Rendering and 3D objects** and **to integrate captured photorealistic 3D textured models in the mixed pipeline**, thus improving the interaction and integration between IBR and the traditional 3D world used in videogames.

New features of the High-Fidelity Prototype

The work performed in Task 4.4 and 4.6 fully implemented the objectives set for this second year of the project, leading to the implementation of a long list of new features and improvements, including (but not limited to):

1. **Unity5 support**: it upgrades the current mixed pipeline to the last major version of Unity, taking advantage of the 64bit architecture and all the new features introduced. Moreover Unity5 has a new licensing model that provides its advanced features (needed by CR-PLAY mixed pipeline) for free to small and independent game development studios. In addition, the prospect of direct inclusion of C++ code in (imminent) future releases of Unity 5 will simplify future integration of new research results.

2. **Development (in WP2) and integration of the faster Bayesian IBR algorithm**: this new algorithm developed since the Low-Fidelity prototype significantly improves the quality of IBR, addressing rendering artifacts and improving speed.

3. **Integration of 3D textured objects from images**: following suggestions from the first review, an extension of the reconstruction algorithms and tools provides photorealistic 3D textured objects ready to be used along with IBR rendering and manually generated 3D models..
4. **Reduction of video-memory and disk occupation**: the refactoring of data-structures, avoidance of duplicated information and data serialization led to an important reduction of occupied space, both in video-memory and disk (particularly important for future deployment on mobile devices).
5. **Improved rendering quality**: a number of glitches, artifacts and noisy issues have been addressed and removed.
6. **Faster loading time for IBR scenes**: pre-computation and serialization of specific data result in the speed up of the initial loading phase, which has significant impact on the acceptance of the new CR-PLAY technology.
7. **Integration of the Reconstruction Tool in Unity Editor**: it integrates the user interface of the reconstruction tool inside Unity, in order to provide a coherent user experience from Capture to Play, responding to user feedback.
8. **Integration of Scale feature in the Reconstruction Tool**: it gives game developers the ability to control the scale of a reconstruction. A feature particularly requested during the first cycle of evaluation.
9. **Integration of Depth Occlusion on IBR scenes**: it features IBR elements of the scene to occlude 3D objects (e.g. characters of a game), by implementing a technique based on the use of the depth information of 3D proxies placed in scene using as reference the point-cloud generated with the reconstruction tool.
10. **C++ version of Depth Synthesis**: this provides a significant speedup in the preprocessing time (from 3-5 minutes/image to 3-5 seconds (!)/image), again responding to user feedback on the time required for preprocessing.
11. **Integration of VBR Semantic Loop**: it extends the current VBR data structure to handle semantics and being able to trigger specific video loops with gameplay events.
12. **Integration of DX11 support**: it removes the previous limitation that prevents the use of the CR-PLAY mixed pipeline outside the OpenGL platform.
13. **Removal of external dependencies**: it removes all external dependencies, (including the use of MatLab in the Reconstruction Tool), and provides a cleaner code base ready to be further refactored toward the full platform independence and the porting to the mobile architecture.

From a purely quantitative point of view, the High-fidelity prototype provides a significant and tangible improvement with respect to the previous version of the pipeline. In particular, considering a typical game made with it, we see:

- A reduction of video-memory usage by around **33%**.
- A reduction of disk occupation by around **43%**.
- A reduction of loading time by around **30%**.
- A reduction of time spent on reconstruction by **40-50%** (depending on scenes).

1. Unity 5

Unity 5 is the most recent major version of the game development tool developed by Unity Technologies [UNITY]. Beyond the numerous features that enrich the new engine release, Unity 5 introduced a new scripting technology called IL2CPP that allows the Intermediate Language (IL - compiled from C#) to be automatically ported to native C++ code before being compiled and executed by the Unity scripting engine.

The introduction of this technology leads to several advantages in terms of performances and scalability, but what is more important for CR-PLAY, will be the possibility (as soon as it will be fully supported) to integrate native code directly inside the Unity's workflow without losing the multiplatform capability provided by the game development tool. This particular aspect will allow the direct integration of native IBR code without creating an external plugin for each supported platform and will give an important advantage in terms of integration effort, especially for future innovations in the rendering algorithms. Besides, it lays the foundation for the design of a more generic IBR SDK that could be easily integrated in different game engines. Figure 1 shows how the CR-PLAY architecture will be improved thanks to IL2CPP.

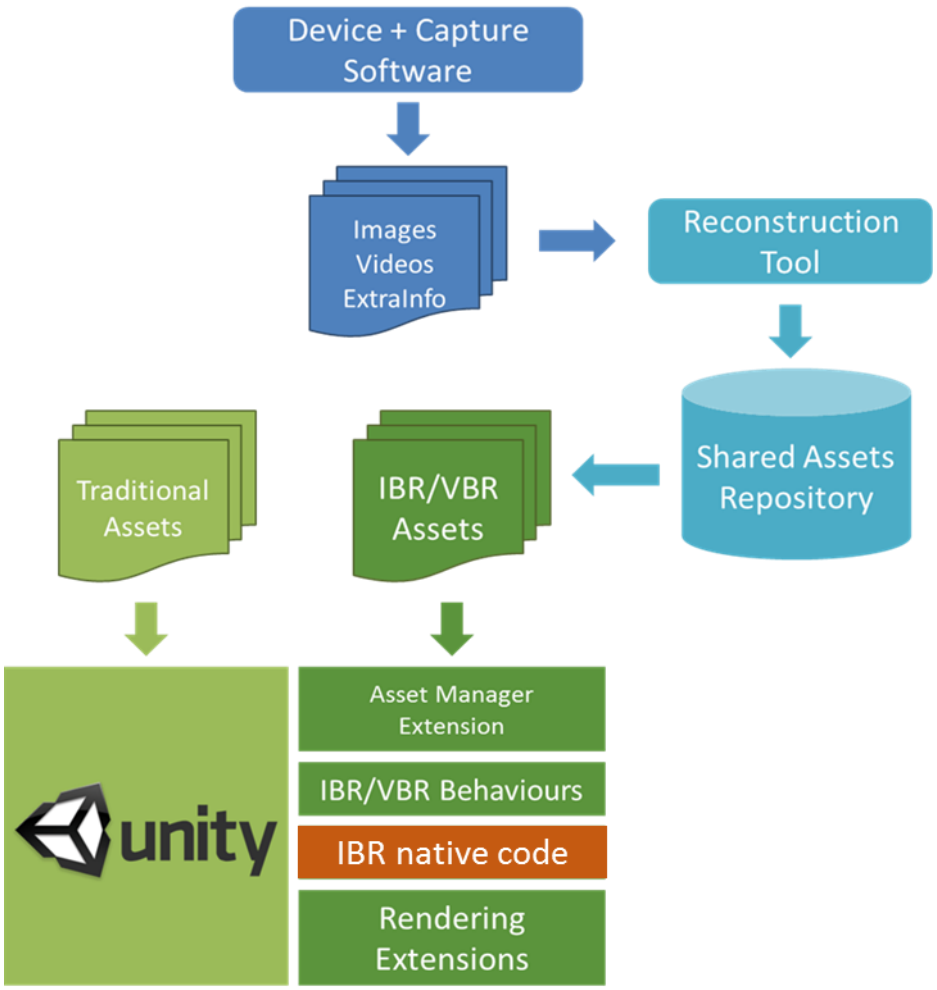


Figure 1 – Future CR-PLAY Architecture modified to handle native code thanks to IL2CPP

Another important feature brought by Unity 5 is the deployment on 64bit platforms. The main advantage is the vast amount of memory that can be allocated by an application (theoretically more than 16 million terabytes). As a consequence, the only limit will be the amount of physical memory present on the machine running the application, with a consistent increase of number of images used for the IBR scene.

Moreover, Unity 5 is now distributed as free-to-use to those legal entities with less than $100,000 of annual gross revenue. This new licensing model is very interesting for individuals and small independent studios and it removes a potential obstacle to the adoption of CR-PLAY technology. It also greatly simplifies development testing for all partners (no need for expensive licenses for all students, postdocs, interns etc.).

The process of porting the CR-PLAY mixed pipeline from Unity 4 to Unity 5 consisted of the following macro steps:

- Building the IBR plugin for the 64bit architecture.
- Including the 64bit version of all the libraries and dependencies.
- Modifying the Unity Scripts following the Unity 5 guidelines [Unity5Guide].
- Refactoring the Importer Tool to organize the dataset following the Unity 5 constrains (i.e. data in the StreamingAssets folder are no longer serializable and directly loadable in the Unity scene).

2. Capture and Reconstruct

2.1 Creation of 3D textured objects using the Reconstruction Tool

An important observation at the first CR-PLAY review was the ability to use the reconstruction technologies of the project to create “traditional” assets using image-based technologies.

In order to provide game developers the possibility to create complex and photorealistic 3D textured models starting from images (potentially captured in the same session of the ones for the IBR rendering), the CR-PLAY mixed pipeline has been extended with the texturing algorithm by Waechter et al. [Waechter 14] developed by TUD (the source code for this technique is freely available on GitHub).

Figure 2 - An untextured mesh generated by surface reconstruction (left) and the final textured object using [Waechter 14] (right)

Figure 3 shows the new reconstruction pipeline extended with the texturing capability.

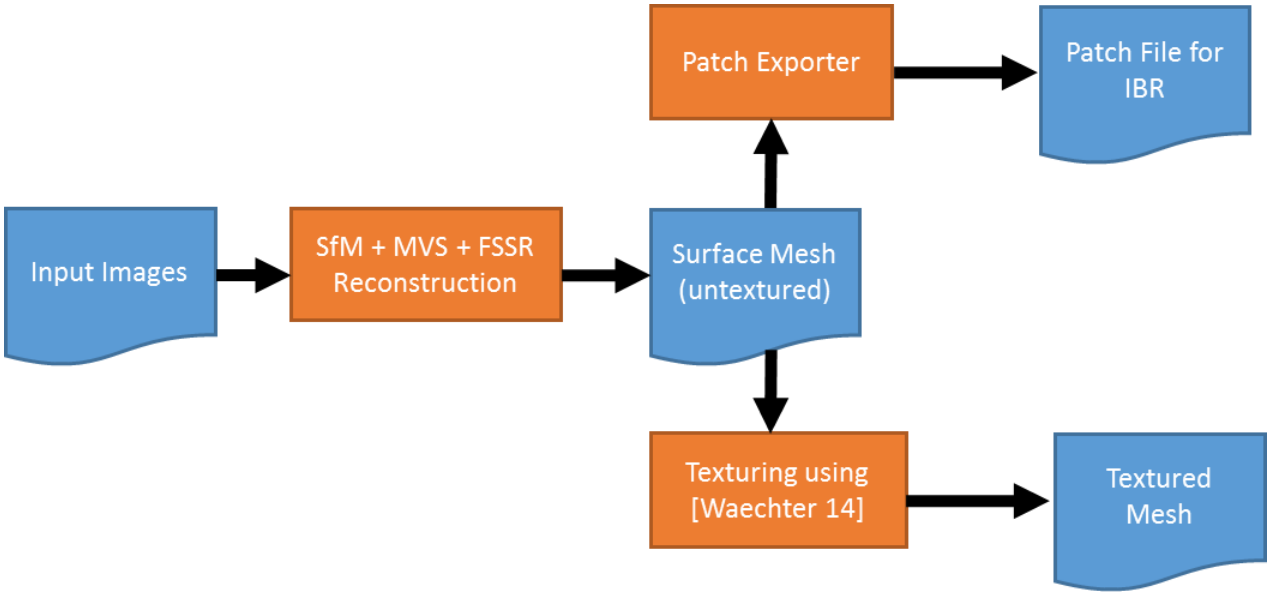


Figure 3 - The new reconstruction pipeline can create textured object using [Waechter 14]

2.2 Integration of the Reconstruction Tool in Unity

To integrate the Reconstruction Tool in Unity and provide a coherent user experience, the orchestration of the reconstruction process has been ported to a set of Unity scripts (C#), removing the use of the external tools used in the Low-fidelity prototype (CMake and Visual Studio). The scripts provide a simple interface (Figure 4) where the user can select the input data, the desired output path, and set several parameter to customize the reconstruction process.

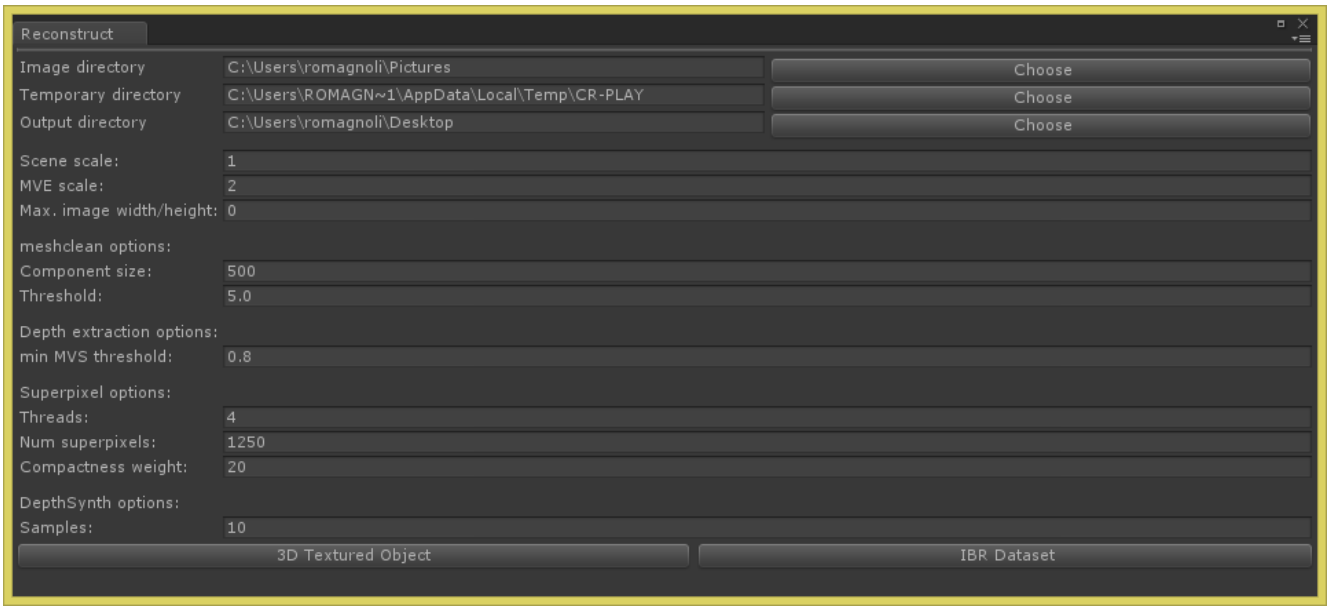


Figure 4 – Unity interface of the Recontruction Tool

The output generated using this Unity tool is a batch file that provides the instructions to execute the reconstruction steps needed to obtain IBR datasets or 3D textured object (depending on the option chosen from the user interface). This greatly simplifies the adoption of CR-PLAY technology by game developers.

2.3 Scaling the reconstruction

The ability to change the reconstruction scale is one of the main feedback items from the end-user evaluation of the Low-fidelity prototype. With the implementation of this feature, game developers have the option to rescale the reconstructed scene setting the "Scene scale" parameter present in the user interface of the reconstruction tool.

After taking into consideration different possibilities, a single parameter option proved to be more effective from both a development and a usability point of view. By setting this parameter, the end-user indicates the distance in meters between the cameras that took the first two images, allowing the reconstruction tool to adjust camera positions and reconstructed 3D points accordingly (by multiplying the scene with a scaling matrix after the Structure-From-Motion process).

2.4 Use of Undistorted Images for IBR

Another feature that helps reduce visual artifacts and improves overall rendering quality is the use of undistorted images for the IBR scene. In these images the radial distortion from the camera lens is removed increasing the quality of IBR renderings.

The distortion is described as a displacement of image points p according to their distance from the image center $r(p)$:

$$p_{distorted} = p_{undistorted} \cdot (1 + \tau \cdot r(p_{undistorted}))$$

The parameter τ is estimated during the Structure-From-Motion step and the undistorted image positions are easily computed afterwards. A simple resampling of the original images with the undistorted positions then leads to an undistorted image. Figure 5 shows an example of an original and an undistorted image. This step is particularly important when using low-end cameras, such as GoPro's or mobile phone cameras.

Figure 5 - The distortion of the camera lens warps points towards the image center. Straight objects in the original image (left) are therefore warped at the borders of the image. The undistortion process removes the image artifacts (right), as shown by the orange reference line.

2.5 Capture Guidance

As part of the work done on the mixed pipeline for the High-fidelity prototype, the capture guidance has been improved by developing a prototype application that support the end-user during the capture phase. The prototype has been developed on the Google Tango [TANGO], a high performance tablet device that allows a real-time rough reconstruction of the captured scene.

More details on prototype and results achieved are reported in D1.2 v.2.

3. Image Based Rendering

3.1 Improvement of rendering quality

Quality improvement for the IBR algorithm was based on the comments and feedback gathered from users and from the various partners of the consortium. We addressed three main issues that were identified as significant visual artifacts:

- Artifacts due to the fronto-parallel assumption of the warping IBR approach [Chaurasia 13]
- Artifacts due to camera selection
- Artifacts due to inconsistent luminance

Fronto-parallel Assumption Artifacts

The warping algorithm of [Chaurasia 13] implicitly assumes that every superpixel is oriented in a fronto-parallel manner, since the warp is performed in the 2D image plane. While this restriction is often not very problematic, it does affect results if we have surfaces which are at an angle (worst case 90 degrees) with the image plane (e.g., floor plane in a scene). In the new Bayesian rendering algorithm (reported in detail in D2.1) we address this issue by fitting local planes to each superpixel; in the cases where the superpixel planes are successfully fitted, we greatly improve the quality of rendering.

Camera Selection Artifacts

For camera selection we tested the following idea based on the intersection of projected camera frusta. It relies on two main features: the **clipped frustum** and the **used frustum**.

The clipped frustum is the screen area occupied from a selected input camera and it is defined as:

$$clipped\ frustum = \frac{clipped\ convex\ hull}{screen\ area}$$
$$(0 \leq clipped\ frustum \leq 1)$$

Figure 6 shows the *clipped convex hull* in red, whereas the *screen area* is the whole rectangle on the right.

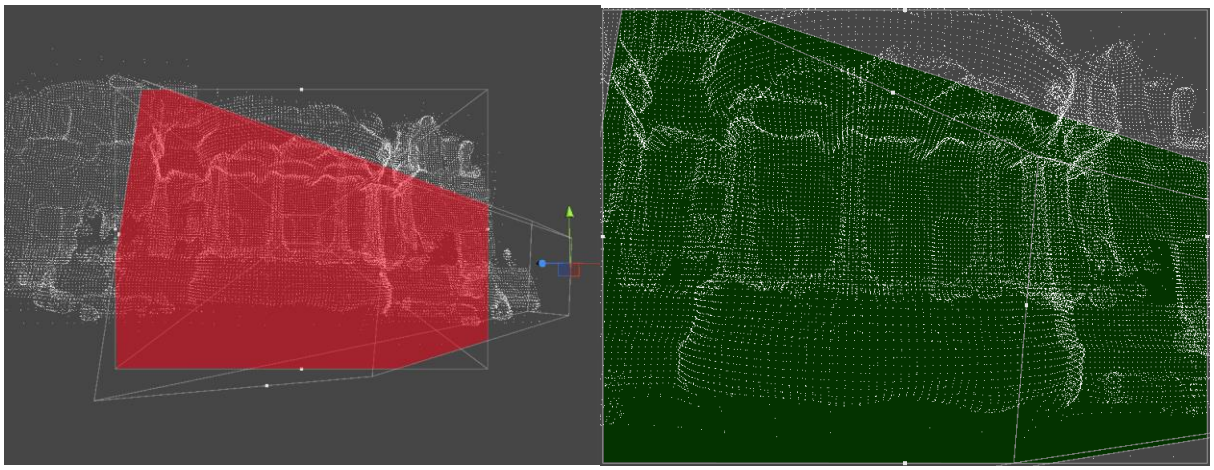


Figure 6- Clipped Frustum: on the left, the red polygon represents the frustum of the input camera projected and then clipped in the 2D camera space. On the right a representation of the clipped polygon in camera space.

On the other hand, the frustum used is the fraction of the input camera projected frustum that is actually clipped in the screen view and is defined as:

$$used\ frustum = \frac{clipped\ convex\ hull}{convex\ hull}$$
$$(0 \leq used\ frustum \leq 1)$$

In the image below (Figure 7) the clipped convex hull is highlighted in red, whereas the whole convex hull is in blue.

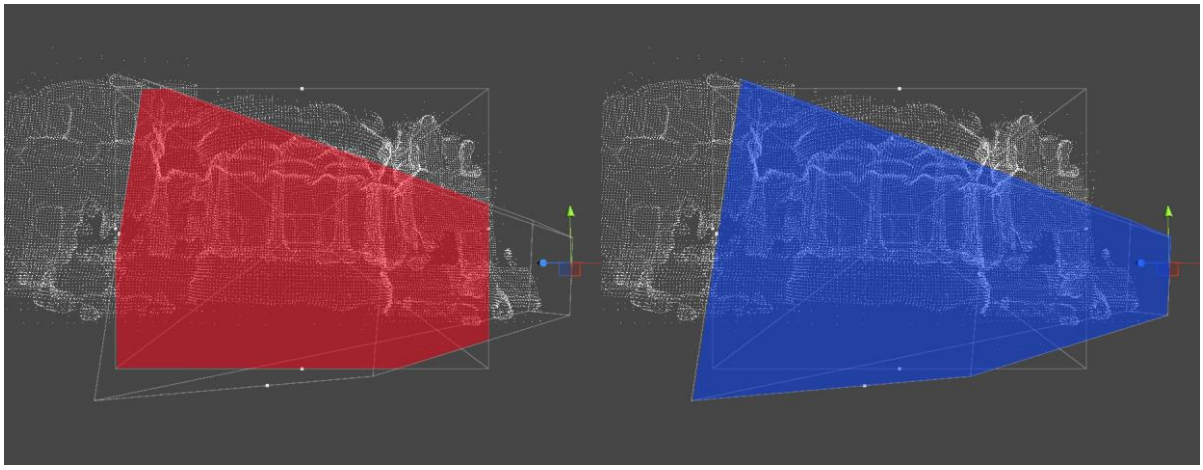


Figure 7 - Used Frustum: on the left, the red polygon represents the frustum of the input camera projected and then clipped in the 2D camera space. On the right, the blue polygon represents the whole projected frustum

In the end, the idea is to select those input cameras that maximize the product of the two previous values.

$$camera\ value = clipped\ frustum \times used\ frustum$$

The conclusion of this test is that we need to work further on this problem in the context of the new Bayesian rendering algorithm. With the new algorithm the ability to treat slanting planes and the increase in speed (reported in D2.1) gives us significant leverage to determine new camera selection approaches, notably due to the fact that we can use more cameras and that we can project accurately from cameras far away if they project onto planes, which was not the case in the original warp-based algorithm.

Inconsistent Luminance.

Even when taking photographs over a short period of time with fixed exposure, illumination does change over the time required to take a few hundred images, i.e., the sun changes position which leads to shifts in shadow boundaries and in overall illumination levels. These changes are small and rather subtle when looking at each individual image, but are visible in the resulting IBR experience since these images are blended together and even small change become visible and annoying. To address these problems, we developed a solution to harmonize colors across the images. The process proceeds as follows.

1. We loop through all the PMVS points in the patch file and store all the colors corresponding to each point in all images in which the points are visible.
2. We take the median color for each point.

3. We then set up a linear system to compute a 3x3 color conversion matrix which will find the best color transformation transforming the color of these points to their corresponding median color in a least squares sense.
4. The solution of this system is the color transformation that we then apply to each image.

Since this is a single global transformation for each input image, in some cases we have colors out of range. In this case we clamp the LAB values to the correct range, which give satisfactory results. It may be necessary to provide special treatment for the sky pixels to avoid residual artifacts, or adopt a more local approach.

Thanks to this process it is possible to correct the small changes caused by the inconsistent luminance and improve the IBR rendering quality, but changing the light information will need significant development work to apply the research results achieved in WP2. Due to the amount of development work done to deliver a stable and reliable High-fidelity prototype, the development work on delighting-relighting feature is still in progress and its integration in the CR-PLAY mixed pipeline has been shifted to Year 3.

3.2 Reduction of the video memory footprint

Video memory was one of the main bottlenecks observed in the Low-fidelity prototype, so the work of this WP in the second year of the project started by performing a memory footprint analysis that shown that the **60%** of the total used video memory was used by the superpixels textures, a set of ARGBFloat textures that, for each pixel of the input images, store the superpixel ids and the three superpixel neighbours ids on 16 bytes (4 bytes for each id). Due to the relevant number of superpixels in an image, as the number of images increase, the video memory consumption grows very quickly.

The first idea to reduce the video memory footprint was to avoid storing the three neighbour superpixels ids allowing data to be stored on smaller RGBA32 textures. This allowed to reduce the video memory used by superpixel textures from 60% to **26%**, but induced a quality loss due to the missing information about neighbours.

The second idea was to compress the superpixels information using a RLE algorithm that would allow to use two RGBA32 textures for each input image to store the compressed information. This approach would allow to reduce the video memory used by superpixel textures from 60% to **27%** without any quality loss, but needed the implementation of a real-time RLE decoder on GPU that is a complex and time demanding task.

The third and final approach solved the issue by implementing a simpler solution, withvery minor quality loss, called **MedDepth Inclusion**. The idea behind the MedDepth Inclusion was to avoid storing all the neighbour superpixel ids, keeping just a smaller value that will allow the IBR algorithm to easily compute neighbours at runtime. This value is the superpixel's **median depth** that is the same value used by the pre-process IBR phase while computing neighbour's superpixels.

Since the median depth is a float value between -1.0 and 1.0, it was possible to store its quantized form in 1 byte, with a quantization error of 0.004 (small enough to get the same precision of the neighbours ids computation in the IBR pre-process). Moreover, the superpixel id was stored on only 3 bytes (reducing the global maximum numbers of superpixels from 4 billion to 16 million) allowing the entire data structure to be stored in 4 bytes (3 bytes for the superpixel id and 1 byte for the median depth) and use one RGBA32 texture, as proposed above, that will reduce the video memory used by superpixel textures from 60% to **26%** withvery minor quality-loss (that translates in **a reduction of 33% of total video memory used** by the

application). The potential quality loss can occur in the case where the warped superpixels create a different set of neighbors compared to the input images. We have not observed this effect visually.

The process to implement the MedDepth Inclusion proceeds as follow:

- Serialize the superpixel's data storing the superpixel id in the first 3 bytes and the quantization of median depth value in the last byte of a RGBA32 texture. Moreover, splitting these values between color and alpha channels allowed the Unity Editor to automatically be able to show superpixel or depth information by simply switching the texture visualization from color to alpha channel and vice versa, which was useful for debugging.
- The modified warp shader masks the superpixel by comparing the median depth information of its 8 neighbors (top-left, top, top-right, left, right, bottom-left, bottom, bottom-right). This is where the error in neighbors can potentially occur.

The direct result of the reduction of used video memory is the quality improvement brought by the possibility of using more images in a specific IBR dataset. For example, in the Silver Arrow scene it is now possible to use **twice** the number of input images on the same target machine and thus improving the rendering quality as shown in Figure 8.

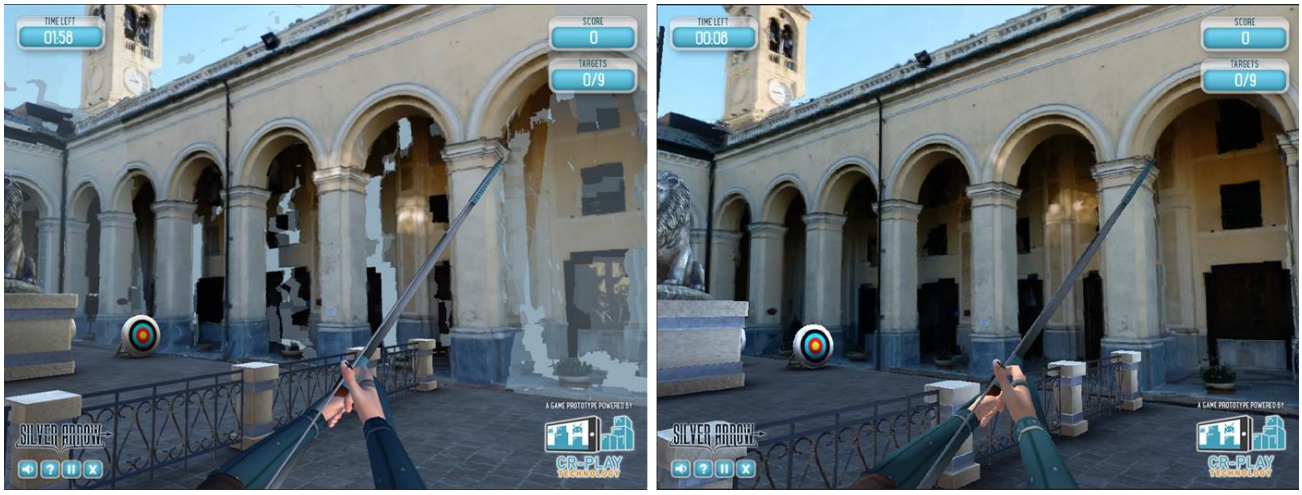


Figure 8 - On the left the Silver Arrow scene rendered with 60 images, on the right the same scene rendered using 120 images.

3.3 Further optimizations of the IBR Plugin

The analysis of the Low-fidelity prototype of the pipeline highlighted three additional major disadvantages related to games based on IBR, namely:

- A considerable usage of disk space.
- A non-negligible loading time.
- Support of the OpenGL rendering platform only.

To deal with the first two problems, a benchmark scene (constituting the Silver Arrow game prototype created in the first year of the project) has been analyzed and consequent solutions have been elaborated.

Silver Arrow scene's elements	% (of disk space occupation)
Unity binaries (EXE and Unity files)	6%
External dependencies (DLLs)	1%
MVS files for depth data-structure (ascii)	1%
SP files for superpixel data-structure (binary)	17%
SP-GRAPH file for superpixel graph data-structure (ascii)	3%
Superpixel textures (RGBA32)	68%
Superpixel graph textures (RGBA32)	4%

Table 1 – Percentage subdivision of the disk space occupation of the Silver Arrow scene.

Table 1 shows the results of the disk space occupation analysis and highlights the elements that take more space on disk, with respect to the entire IBR scene, providing an indication on the priorities of the optimization interventions.

Reduce the size of Superpixel textures – The MedDepth Inclusion approach descripted above (implemented to save video-memory during runtime execution), automatically shrinks the data-structure and reduces the disk space needed by superpixel textures by 75%.

Remove the SP-GRAPH file and reduce the size of SP files – The current format of these data-structure contains duplicate information, already stored in the superpixel textures. The data-structure refactoring allows to remove the SP_GRAPH file and reduces the SP file to a non-significant size (less than 1MB for all the input images) without any quality loss.

Reduce the size of Superpixel graph textures – During the analysis, additional test were made to check if current data-structure was designed in an effective way. Superpixel graph was stored on 1024x1024 textures allowing to contain more than 3 million nodes, but considering a typical IBR dataset, the number of nodes is smaller and most of the texture space was not used. Superpixel graph textures now contains only the significant data reducing the space needed without any quality loss. (i.e., in Silver Arrow scene, the texture size has been reduced by more than 75%).

Get rid of External dependencies – Integration of the Eigen C++ template library [EIGEN] and removal of the SuiteSparse linear algebra package [CHOLMOD] to implement the Cholesky decomposition. This operation removes the need of any external dependency.

Serialize Eigen sparse matrices and warp meshes – At the beginning of the IBR process, the warping mesh (and the associated linear system) is created for each superpixel of each input image. This operation is

time demanding and so it has been moved to a pre-process phase where the warped meshes and the Eigen sparse matrices are serialized to a file to be loaded at the beginning of the IBR process. Thanks to this simplification the loading time of the Silver Arrow scene has been reduced by **30%** without any quality loss on the final result.

The third problem introduced by the use of the IBR plugin is the support limited to the OpenGL platform. The reason of this limitation is the use of GLSL as language to write shaders needed by the IBR algorithm. To make the algorithm able to take advantage of the multiplatform capabilities of Unity, all shaders have been translated in the ShaderLab language that is automatically ported to the target platform by the deployment procedures of Unity, allowing, for example, to run the Silver Arrow scene on the modern DirectX 11.

Summarizing, the optimization phase allowed to **improve the entire disk space occupation (of a game) by around 43%**, to **speed-up the loading phase by around 30%** and to **provide a wider compatibility** towards all rendering platforms supported by Unity.

Beyond the general improvement, the IBR optimization makes an important step towards the porting of the IBR technology on mobile devices, where the limited amount of memory and the low performance play a relevant role.

3.4 Integration of the IBR Bayesian algorithm

As introduced above, the **Bayesian algorithm** (described in detail in D2.1) improves the previous **Spixel Warp algorithm** by introducing the possibility of selecting a different rendering approach, for each superpixel, between following options: warp, planar and front-planar.

From the integration perspective, the Bayesian algorithm extends the current approach on both data preparation and rendering phase.

The data preparation phase is extended by adding the data structures that describe the planar regions and the rendering approach for each superpixel of each input image. This is integrated as an additional step of the Reconstruction Tool pipeline.

In the rendering phase, the planar data coming from the data preparation phase are used to update the rendering data structure with the selected rendering approach and to create the additional plane for each superpixel of each input image. Thanks to these data, Unity is able to orchestrate the rendering pipeline by calling the new IBR feature implemented in the native plugin.

The integration of the IBR Bayesian algorithm provides improvements on the final rendering quality by fitting local planes to superpixels that avoids the fronto-parallel assumption and potentially allows the use of more than four input cameras to compute a more precise novel view from the virtual camera.

3.5 Depth Occlusion

One of the interesting features for game developers dealing with CR-PLAY pipeline is the possibility of having parts of IBR scene occlude traditional 3D objects. To achieve this, it is possible to use the depth information inside the superpixels and render them on different depths in order to let them occlude the objects in the 3D world. Despite of the simplicity of these rendering operations, this approach suffers

from evident artifacts caused by two different problems: low accuracy on the depth information associated with superpixels and low accuracy on superpixels' segmentation (Figure 9).

Figure 9 - Superpixel depth artifacts

Both are complex problems that need a significant research and development effort to be solved, which is currently under way. After analysis, a simpler and effective way to simulate the depth information in the traditional 3D world has been elaborated.

The Depth Occlusion bases its operating principle on the use of 3D proxy models. Despite the effort needed to create proxies is directly proportional to the shape's complexity, depth proxies can potentially approximate any shape and provide precise information about object depth in specific scenes. Figure 10 provides an example of how the system works.

Figure 10 – The proxy-column and the 2 characters are the only 3D objects in the scene. 3D objects are placed in the space using the spatial reference provided by the point-cloud in the Unity Editor.

In the example, the column's models have a dedicated shader that tells the rendering pipeline to consider only their depth information (making them invisible), and brings the IBR backdrop on top of other objects using their accurate depth. Additionally, those models that need to be "masked" by the IBR scene need to have a specific behavior attached that allows them to be moved on the proper position in the rendering queue. The final result can be seen in Figure 11.

Figure 11 – Characters are rendered taking into account the depth information provided by the proxy-columns, allowing the IBR background to be rendered on top of them where needed.

This approach simulates the depth information of the IBR scene by using the point cloud as reference to place the depth proxies in the 3D world. Since this method is completely decoupled from superpixels, it

is not affected by the problems explained above. Future research will eliminate this problem, allowing proper depth interaction for various shapes in the scene; evidently some cases are particularly hard, such as thin structures like fences, and will require specific treatment.

3.6 Porting of Depth Synthesis to C++

The depth synthesis code was a significant bottleneck in the Low-fidelity prototype, since the implementation was done in MatLab. This is a complex piece of code, with several interdependent modules and functions involved. To convert the code to C++, we first identified software libraries suitable to replace those in MatLab, in particular we chose to use *OpenCV* [OPENCV] for image and color handling, and *boost* [BOOST] for its tree/graph representation and its Dijkstra algorithm implementation.

The conversion process started by translating the MatLab functions to C++ and building the data structures for the tree structure used to propagate depth.

We wrote three main classes:

- *DepthSynthesize*, which groups and manages the main functionalities: reading data, identifying source and target pixels, setting up the graph/tree structure, performing the propagation step, creating median depth and saving data;
- *DepthSynthesisUtil*, which performs the actual synthesis and saves the median depth and the new mvs files;
- *SpGraph*, which encodes the graph/tree structure used for the propagation step and provides the wrapper for the Dijkstra algorithm functionality of boost.

Several difficulties were encountered in the development of the C++ solution, notably in the different way C++/MatLab handles color histograms and their chi-square comparisons, color conversions and initial binning for depth, including for non-synthesized pixels. This was not well documented in the initial MatLab code, and required extensive debugging to become operational.

Despite of the technical difficulties, the native implementation of the current Depth Synthesis brought a significant improvement by taking **only 2-3 seconds per image** to perform this task (the previous MatLab implementation took about 5 minutes per image). This provides a resulting speedup of over the 100x and the significant decrease of time needed by the end-user to get a working IBR dataset from the captured images. As an example, the depth synthesis of the Silver Arrow scene (60 images) took about 4 hours in the Low-fidelity prototype, whereas it takes only 2 minutes in the High-fidelity prototype.

4. Video Based Rendering

4.1 Integration of the Semantic Loop in Unity

In the Low-fidelity prototype, VBR was integrated by loading each video-frame of a video loop and playing it from the beginning to the end, simulating a moving object with a realistic look and feel. In the High-fidelity prototype, VBR has been expanded to make the technology able to switch between video loops when specific events are triggered from the gameplay.

To achieve this result, VBR data-structures have been expanded by representing each video-frame as a node of a directed graph structure (Figure 12).

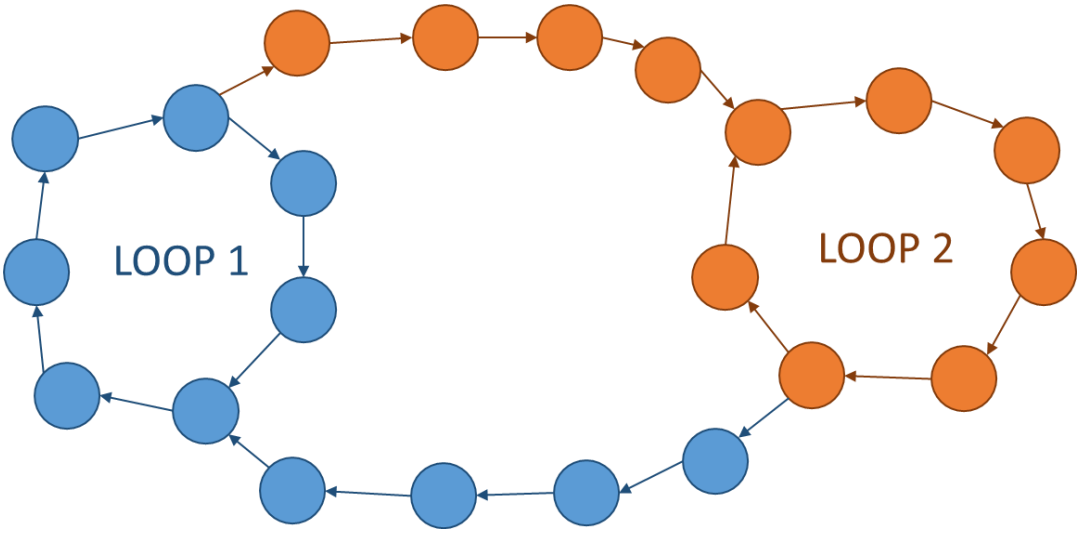


Figure 12 – Example of a direct graph representing the VBR frames and their relationship. Different colors means different semantic labels. Loops own their incoming branches.

The graph is structured in connected loops where each loop represents a video-loop connected to other loops by branches. Thanks to this data-structure the video-texture can switch between different loops by simply following the link to the next node until a branch to another loop is found.

In terms of integration, the VBR pre-processing pipeline produces a set of video-frame textures and a file containing the definition of the graph. These files are serialized in Unity thanks to the extended importer tool (Figure 13).

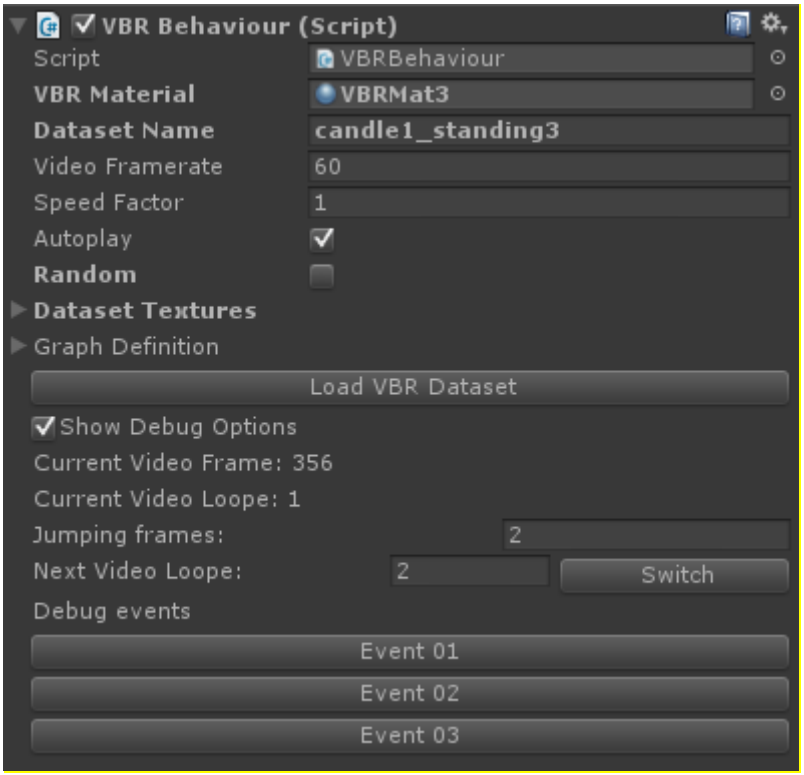


Figure 13 - Snapshot of the VBR Unity Editor interface

Once VBR data are loaded in Unity, the VBRBehaviour component is responsible to take the video-frame textures and the graph definition and manage the VBR objects by reacting to external events triggered by the gameplay.

Example of use: the candle

To explain the use of the VBR Semantic Loop in Unity, we will consider the example of a candle moved by the wind.

The candle has three different behaviors:

- Straight: the candle stays straight, i.e. no wind is present.
- Left: the candle leans to the left, i.e. a wind from right is applied.
- Right: the candle leans to the right, i.e. a wind from left is applied.

Each of three behaviors is represented by a video-loop and each loop is connected to each other, in both directions, with one or more connection branches (Figure 14).

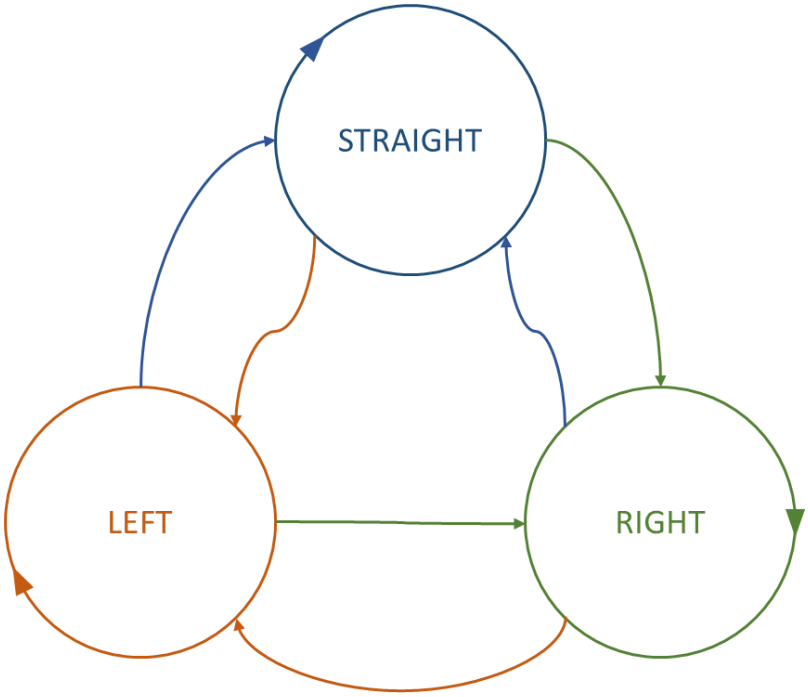


Figure 14 - Representation of video-loops in the candle example

Once the video-frame textures and the graph definition are loaded in Unity by the VBR importer, VBRBehaviour has all the information to play the available loops.

The straight loop is played by navigating the graph, following the next node with the blue semantic label. Once the gameplay triggers the “wind from right to left” event, the VBRBehaviour catches the event and looks for the next node with the orange semantic label. When the node is found, the graph navigation continues following the next orange node and playing the frames in the connection branch and then the frames in the left loop.

Since all loops are multiple-connected between each other, it is always possible to go from one loop to another and play the candle video loop accordingly to the current gameplay behavior.

5. Risks and contingency plan

Risks and contingency actions table				
Id	**Risk description**	**Probability (low, med, high) - Comment**	**Impact (low, med, high) - Comment**	**Contingency action**
Risk 4.1	Camera selection work will not provide improvements on the rendering quality.	**Med/High** – it is still not completely clear how to implement the new selection approach.	**Med** – improvements on quality will be lower than expected.	In the worst case, we will be able to simply use more cameras which will improve the quality in most cases.
Risk 4.2	IBR Bayesian algorithm will not work as expected when used to create game prototypes.	**Low –** testing is not complete, but the algorithm has been used on several scenes in a research context.	**Med** – improvements on quality will be lower than expected. However the speed increase is significant.	The end-user will be able to switch to the previous Spixel warp algorithm.
Risk 4.3	Game developers could provide disruptive feedbacks on usability and technical choices taken during the development of the High-fidelity prototype	**Low/Med** – design and development choices have been guided by feedback from game developers and market orientations. However visual artifacts remain; these will be addressed with further research in Year 3.	**Med**- additional work will be required to modify the pipeline toward the Final prototype.	User feedback will be addressed in the development of Final prototype.

6. Conclusion

The High-fidelity prototype of CR-PLAY pipeline represent a major advancement with respect to the previous release, in terms of new features, improvements, optimizations and quality. Most of the research achievements of WP1, 2 and 3 are now part of the pipeline, with a coherent user interface. Game developers are now able to create better and richer games, using Unity 5 as main development tool.

6.1 Plan for next period

Following the three-tier schema that defines development tasks in WP4 (Low-fidelity, High-fidelity and Final prototypes of mixed pipeline), the next step is to collect feedback coming from MS14 "Results of High-fidelity prototype evaluation and recommendations for next design iteration" (WP5), so as to organize and prioritize the integration and development tasks that will lead to the creation of the Final prototype in Year 3.

In addition, research continues in the development of the new algorithmic solutions for IBR and VBR; some of these (better depth synthesis, semi-automatic removal of distracting objects in input images, multi-view VBR for vehicles, handling of indoor scenes, rendering of thin objects, better render of “midground” objects such as parked cars), may be integrated in the final prototype, while others will remain research proof-of-concepts. Any of the above advancements will result in significant improvement in visual quality and the overall CR-PLAY experience.

Partners involved in development WPs will have specific roles, and the work will be performed according to the plan developed first in the DoW and then in D4.2. More detailed activities will be defined within sub-groups and depending on specific needs within the shell of main tasks of the WPs, towards the achievement of the main project objectives.

References

[BOOST] http://www.boost.org/

[Chaurasia 13] *Chaurasia, G., Duchene, S., Sorkine-Hornung, O., &Drettakis, G.* (2013). Depth synthesis and local warps for plausible image-based navigation. *ACM Transactions on Graphics (TOG), 32(3), 30.*

[CHOLMOD] http://faculty.cse.tamu.edu/davis/suitesparse.html

[EIGEN] http://eigen.tuxfamily.org

[OPENCV] http://opencv.org/

[TANGO] https://www.google.com/atap/project-tango/

[UNITY] http://unity3d.com/

[Unity5Guide] http://docs.unity3d.com/Manual/UpgradeGuide5.html

[Waechter 14] *Michael Waechter, Nils Moehrle, Michael Goesele*, Let There Be Color! Large-Scale Texturing of 3D Reconstructions, In: *Proceedings of the European Conference on Computer Vision (ECCV 2014), Part V, LNCS 8693, pp. 836–850, 2014.*

Annex A – High-fidelity Prototype: User Manual 1.0

1. CR-PLAY Technology in a nutshell

CR-PLAY is an innovative mixed pipeline for videogame development that allows the integration of Image Based Rendering, Video Based Rendering and 3D Textured Objects generated from images, together with traditional 3D contents, in the videogame development process. The pipeline is composed by three main phases:

- **Capture**: the User takes pictures of the scene/object that he wants to insert in the game.
- **Reconstruct**: the User use specific tools to create IBR Assets, VBR Assets and 3D Textured Objects generated from images.
- **Edit and Play**: reconstructed assets are integrated with traditional contents and the game is created and deployed.

Each asset created with the CR-PLAY mixed pipeline have specific characteristics that can be exploited in the creation of a game.

IBR Assets are detailed representations of outdoor and indoor environments, created from a set of input images. They are mainly suited for advanced backdrops thanks to their intrinsic parallax and the possibility of simulating occlusion on traditional 3D objects.

Figure 15 – Images taken during the Capture phase (above) and example of IBR scene (derived from the images) with occlusion from different views (below)

VBR Assets allow the detailed simulation of dynamic elements (moving and animated objects) and represent them in advanced video-textures, able to reproduce specific video sequences according to external gameplay events.

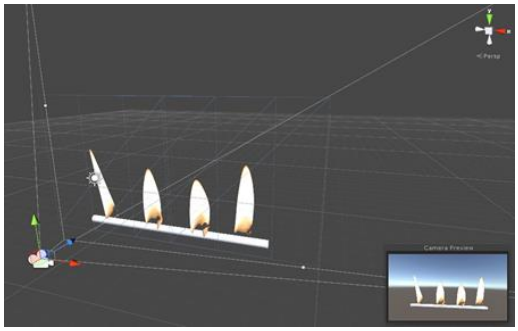


Figure 16 – Example of VBR assets (moving candles) in the Unity Editor

3D Textured Models are traditional polygon models with high-res photorealistic textures, captured and created from a high number of images. They can be used as traditional assets and provide a full interaction with the 3D world.

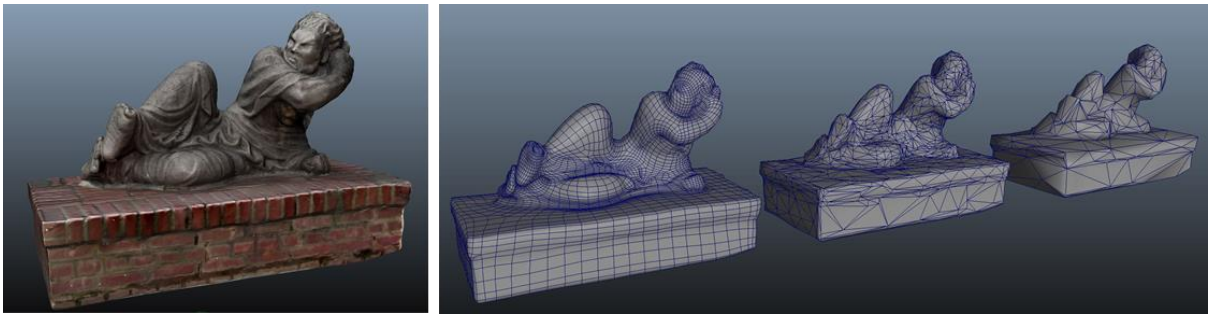


Figure 17 - 3D Textured Model (left) with different optimization levels (right)

Next image (Figure 18) introduces the main steps characterizing the use of the CR-PLAY Technology (next sections of the document provide a detailed explanation of each step).

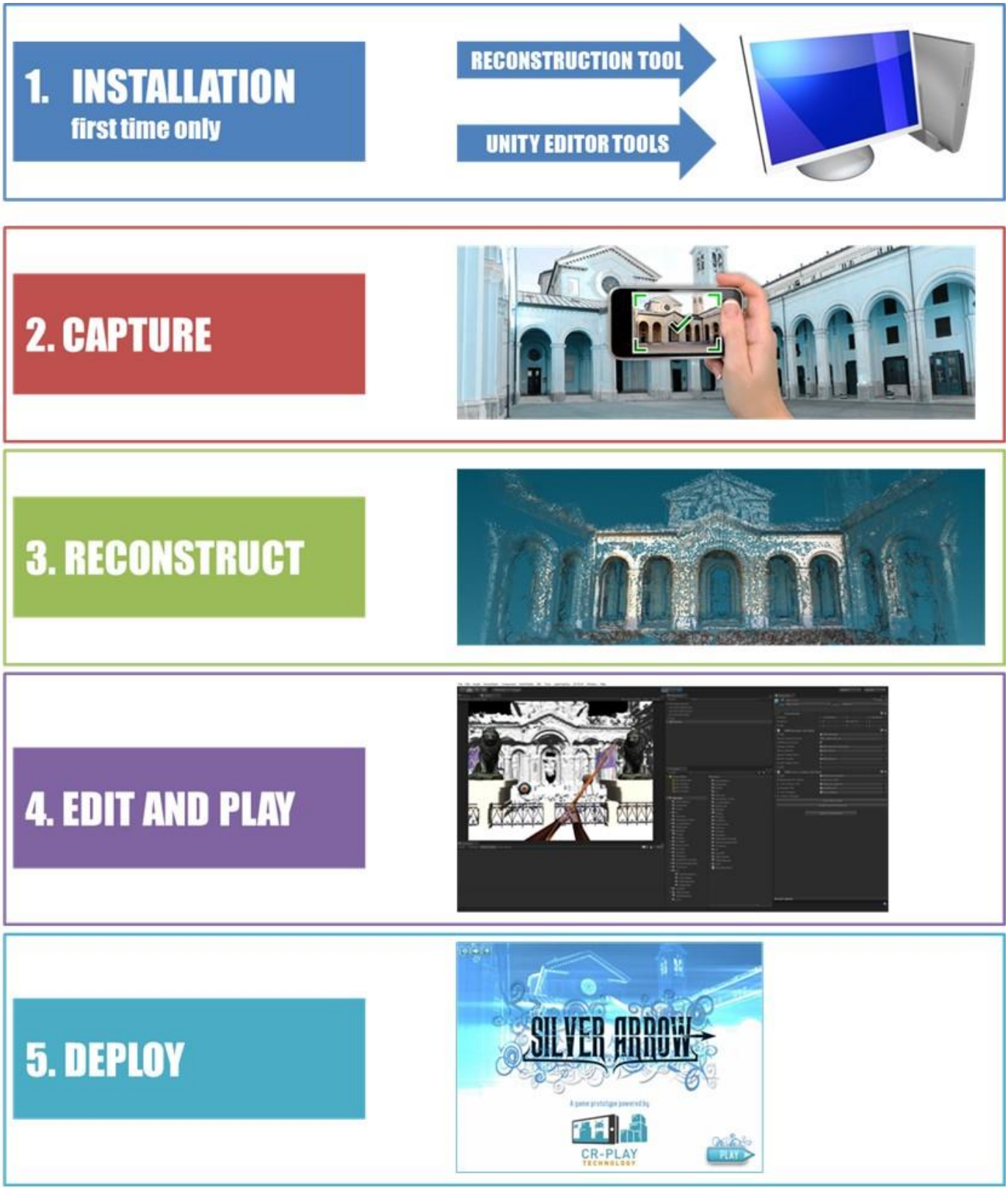


Figure 18: CR-PLAY Use Case schema

2. CR-PLAY System Requirements

The features listed in this section are intended to describe the target developer machine, which will allow the CR-PLAY mixed pipeline to work smoothly. Less powerful machines can give unexpected errors and hinder user experience, especially in configurations with less video memory than the value suggested.

- Processor: Intel i5 4590 3.3 GHz or higher
- Memory: RAM 8GB or higher
- Video card: ATI Radeon R9 200, 3GB VRAM or NVIDIA equivalent or higher

CR-PLAY needs Unity 3D 5.x version.

3. Installation of the CR-PLAY mixed pipeline

In this section the User will install the Reconstruction Tool and all Unity components needed to use CR-PLAY Technology to create games.

3.1 Setup of Unity project and installation of CR-PLAY packages

3.1.1 The User creates a batch file in the project's root folder, in order to start CR-PLAY Unity project in DX11 mode.

Example of batch command:

```
"<path_to_Unity_Editor>/Unity.exe" -force-d3d11 -projectPath %~dp0
```

✔ By running the batch command Unity will start in DX11 mode.

3.1.2 The User opens Unity 5 using the batch file created (double-click on it), sets building parameters and game resolution, and creates the "IbrProxy" layer.

✔ Unity is ready to be used.

3.1.3 The User downloads the CR-PLAY Reconstruction Tool Unity Package from http://www.cr-play.eu/download_sw/ and installs it in the Unity project (username and password required).

✔ Unity is ready to use Reconstruction Tool features.

✘ If the download is not available due to server or credentials problems, the User can contact support@cr-play.eu to get assistance [Error 1].

3.1.4 The User downloads the CR-PLAY Unity Package http://www.cr-play.eu/download_sw/ and installs it in the Unity project (username and password required).

✔ Unity is ready to use IBR features.

✘ CR-PLAY Unity package cannot be installed. Unity provides a very detailed output log that can be used in order to fix specific issues, if he/she cannot fix the issue. He/she could contact *support@cr-play.eu* to get assistance [Error 2].

4. Use of CR-PLAY mixed pipeline for videogame creation

This section explains how to use CR-PLAY technology to create games. Various phases are described: Capture, Reconstruct, Edit and Play, Deploy on Windows platform.

4.1 CAPTURE the scene

4.1.1 The User takes pictures of the scene following the provided capture guidelines.

See instruction manual, page 47: Guidelines for capturing datasets

✔ The scene has been captured.

4.1.2 User copies the pictures from the camera memory to the computer where the Reconstruction Tool is installed.

✔ Pictures are available for Reconstruction.

4.2 RECONSTRUCT the scene

4.2.1 User opens Unity and click on the “CR-PLAY/Reconstruction Tool” menu item, sets the “Image folder” and creates the batch file.

See instruction manual, page 37: How to use the Reconstruction Tool in Unity

✔ The batch script is present in the temporary folder.

✖ The batch file is not present. Unity can show errors in the console that user can use in order to fix specific issues. He/she could contact *support@cr-play.eu* to get assistance [Error 3].

4.2.2 User runs the batch script from the command prompt and waits for the process to finish correctly.

✔ After a while the dataset has been generated in the output folder. NOTE: this can take up to several hours, so please plan ahead

✖ Reconstruction process fails, the User can contact support@cr-play.eu to get assistance [Error 4].

4.3 (EDIT and) PLAY the scene - IBR

4.3.1 User moves the "<ibr_dataset>.zip" file generated in the Reconstruction Tool output folder to the "Assets/IBRDatasets" folder.

✔ IBR scene is ready to be imported.

4.3.2 User selects the "CR-PLAY/Import IBR" menu item.

See instruction manual, page 40: How to use CR-PLAY mixed pipeline in Unity

✔ ImportDataset window opens.

4.3.3 User checks the <ibr_dataset> item and clicks on "Import" button, waiting until the importing is completed.

It can take some time depending on the machine and on the number of images in the dataset.

See instruction manual, page 40: How to use CR-PLAY mixed pipeline in Unity

✔ IBR dataset is imported in the project.

4.3.4 User instantiates the IBRscene prefab in the scene and selects it in order to open its properties in the Inspector window.

✔ Unity is ready to show the IBR scene in its scene view.

4.3.5 User sets the text field "Dataset Name Folder" to <ibr_dataset>, checks the "VSFM Coord Sys" and clicks the "Load IBR Scene" button to load the IBR in the Unity scene.

See instruction manual, page 40: How to use CR-PLAY mixed pipeline in Unity

✔ IBR scene is loaded in the Unity scene.

✖ Unity can raise an error in case the "Dataset Name Folder" is not set to a valid folder or if the Import operation (step 4.3.3) has not yet been done. For any other case, the User can contact support@cr-play.eu to get assistance [Error 5].

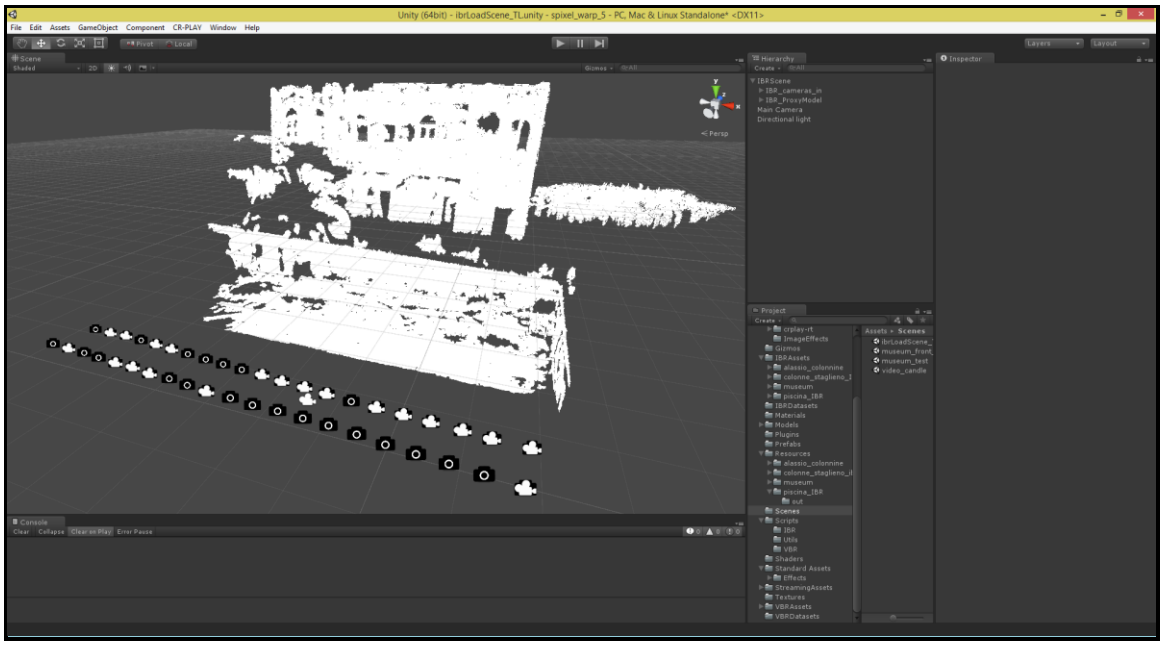


Figure 19 - The IBR scene is loaded in Unity

4.3.6 [Only the first time a specific dataset is imported or modified] User clicks on "Prepare IBR Scene" in order to create superpixel intermediate textures and all needed data.

See instruction manual, page 40: How to use CR-PLAY mixed pipeline in Unity

- Intermediate superpixel textures and other data are generated and placed in "Assets/Resources/", "Assets/IBRAssets/" and "Assets/StreamingAssets" folders. This operation can take time depending on the machine.

- Unity can crash if, due to the high number of input images, the texture generation uses all the available video memory. In this case the User can contact support@cr-play.eu to get assistance [Error 6].

4.3.7 User PLAYS the Unity scene.

- Unity scene is played and the IBR scene is shown after a loading phase.

- Unity scene can give errors depending on the specific gameplay code. Unity provides a very detailed output log that the User can use in order to fix specific issues, if the User cannot fix the issue and it is related to CR-PLAY, he/she can contact support@cr-play.eu to get assistance [Error 7].

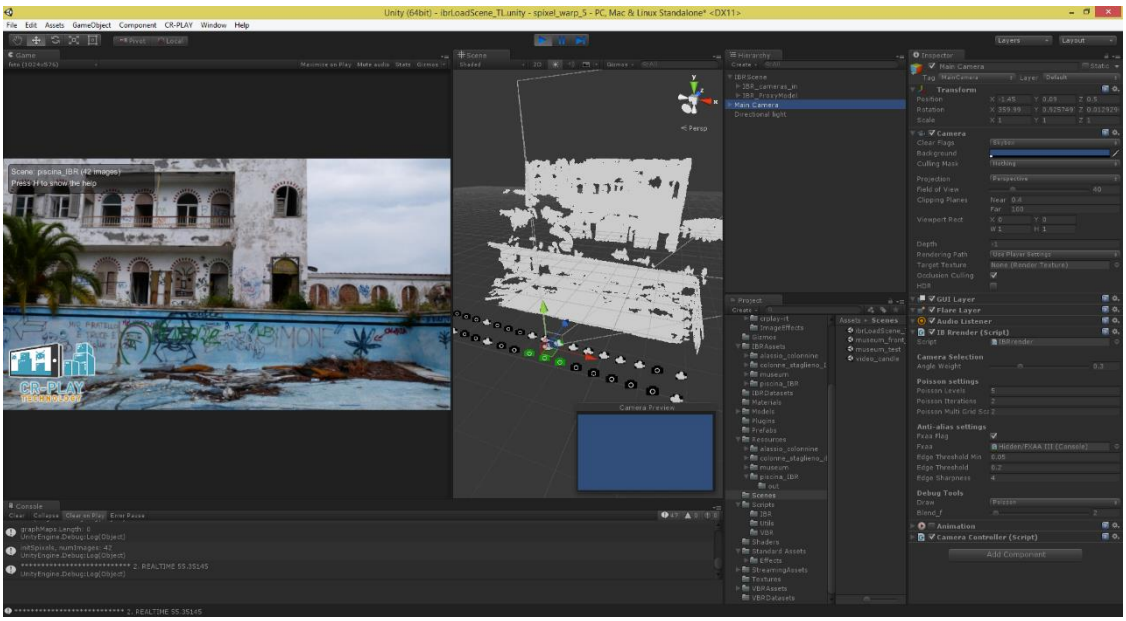


Figure 20 - The IBR scene is played in Unity

4.4 (EDIT and) PLAY the scene – VBR

In case you plan to use animated content in your game that could be done with the VBR technology, please contact support@cr-play.eu to receive assistance.

4.4.1 User moves the <vbr_dataset>.zip in the "Assets/VBRDatasets" folder.

- VBR scene is ready to be imported.

4.4.2 User moves the focus on Unity 5 and selects the "CR-PLAY/VBR Import" menu item.

- ImportDataset window opens.

4.4.3 User checks the <vbr_dataset> item and clicks on the "Import" button, waiting until the importing has been completed.

- VBR dataset is imported in "Assets/VBRAssets" folder.

4.4.4 User instantiates the VBRObject prefab in the scene and selects its child "VideoTexture" in order to open its properties in the Inspector window.

- Unity is ready to show VBR object in its scene view.

4.4.5 User sets the textfield "Dataset Name" to <vbr_dataset> and clicks the "Load Dataset" button to load the VBR video textures.

- Video textures are loaded in the scene.

- Unity can raise an error in case the "Dataset Name" is not set to a valid folder or if the Import operation (step 4.4.3) has not yet been done. For any other case, User can contact *support@cr-play.eu* to get assistance [Error 8].

4.4.6 User sets playback parameters depending on his/her needs.

- VBRObject is ready to be played.

4.4.7 User PLAYS the Unity Scene.

- Unity scene is played and VBR object is shown.

- Unity scene can give errors depending on the specific gameplay code. Unity provides a very detailed output log that User can use in order to fix specific issues, if User cannot fix the issue and it is related to CR-PLAY, he/she can contact *support@cr-play.eu* to get assistance [Error 9].

4.5 Deploy the game

4.5.1 The User clicks "Edit/Project Settings/Graphics" menu item and adds shaders in "Assets/Shaders" folder in the "Always Included Shaders" list.

✖ IBR shader will be linked to deployed packet.

4.5.2 The User clicks on "File/Build Settings" menu item.

✖ Building Settings window appear.

4.5.3 The User clicks "Build" button, select the final executable location and start building.

✖ Building process is started. NOTE: Only a Windows build target is supported for now.

✖ The building process can fail while compiling the scripts due to code errors. Unity provides a very detailed output log that User can use in order to fix specific issues, if User cannot fix the issue and it is related to CR-PLAY, he/she can contact *support@cr-play.eu* to get assistance [Error 10].

4.5.4 User creates a batch file in order to start the created exe in DX11 mode.

✖ The batch file is created.

4.5.5 User double-clicks on the batch file and starts the game.

✖ The game runs showing CR-PLAY technology.

✖ Deployed game can crash for several reasons. Unity deployed applications provide a very detailed output log that the User can use in order to fix specific issues, if the User cannot fix the issue and it is related to CR-PLAY, he/she can contact *support@cr-play.eu* to get assistance [Error 11].

Annex B – High-fidelity Prototype: Technical manual

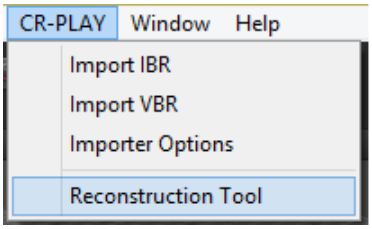
How to use the Reconstruction Tool in Unity

Installation

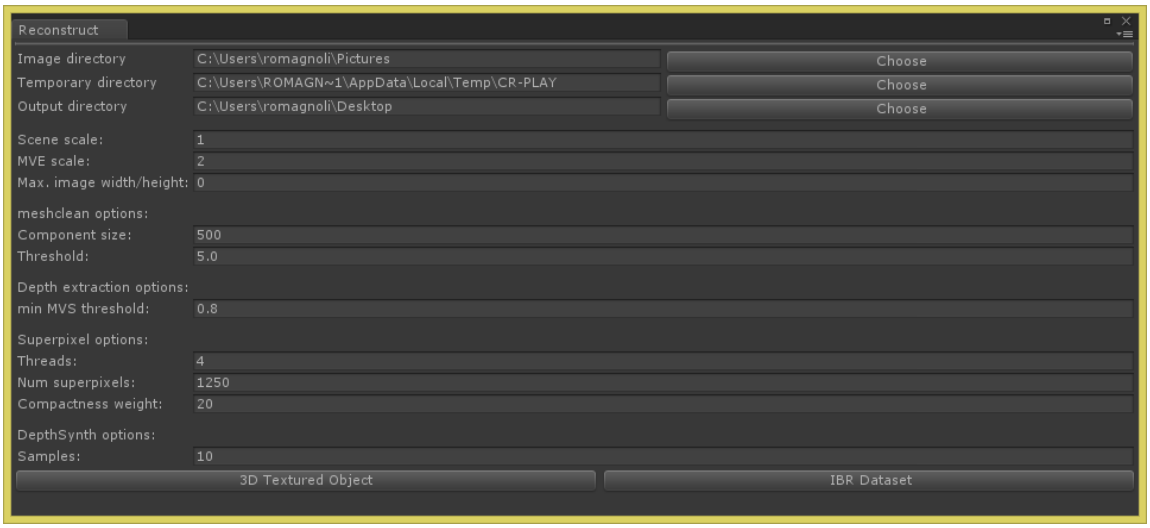
To install the Reconstruction Tool in Unity, it is necessary to import the unity package **`cr-play-reconstruction-tool.unitypackage`**, that automatically installs all needed resources in the selected Unity project.

Usage

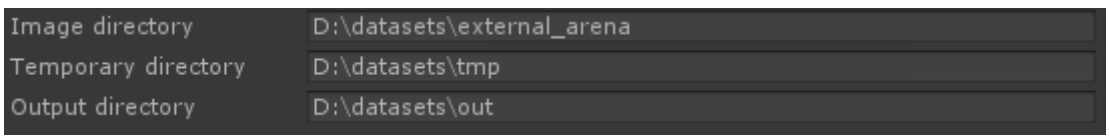
- From Unity click on the "CR-PLAY / Reconstruction Tool" menu item.



- The Reconstruction Tool window will appear presenting the options to the user.

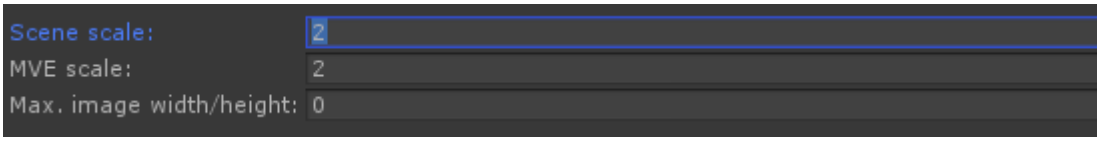


- The user sets the "Image directory" field with the path of the folder containing the images captured and the "Temporary directory" and "Output directory" with the path to the temporary folder and the folder where the output archive will be copied respectively.



- The user sets the "Scene scale" field to set the final size of the reconstruction. **The scale value is indicated as the distance in meters between the first two pictures of the dataset.** Modifying this value will allow the user to scale the size of the resulting reconstruction up or down. Example: setting

the scale value to 2, will provide a reconstruction where the cameras that took the first two pictures are at 2 meters from each other.



- Other fields can be left as they are.
- Once all needed parameters are set, the user can proceed by clicking the "3D Textured Object" button to create a traditional 3D textured model (typically for an isolated object for which photos have been taken all around) or by clicking the "IBR Dataset" button to generate the IBR dataset (typically for backdrops).

Once one of the buttons is clicked, a batch script is created inside the "Temporary Directory".
To run the reconstruction process the batch file needs to be executed from the command prompt.

In the table below you we show the description of all the parameters available through the Reconstruction Tool interface and their explanation.

FIELD NAME	DESCRIPTION
Image directory	Path of the folder that contains the captured images.
Temporary directory	Path of the temporary folder used by the different steps of the reconstruction pipeline.
Output directory	Path of the folder that will contain the output archive.
Scene scale	Scaling factor applied during the reconstruction pipeline (i.e., the distance in meters between the cameras that took the first two pictures of the captured image set).
MVE scale *	Scale used for the whole reconstruction.
Max. image width/height *	Maximum pixel size the captured images will be resized to. 0 means no resizing. Use only even values, since otherwise some of the IBR steps won't work properly.
meshclean options: Component size *	Minimum number of vertices per component on the mesh reconstruction step.
meshclean options: Threshold *	Threshold on geometry confidence: N in a range between 0 and the image number. Takes into account 3D points for surface recognition only if visible from N cameras.

Depth extraction options: min MVS threshold *	The minimum MVS point confidence threshold, useful for back-projection. Try 0.0 if black depth-maps are generated.
Superpixel options: Threads *	Number of threads used to compute superpixels.
Superpixel options: Num superpixels *	Defines the number of superpixels you want for the oversegmentation's step.
Superpixel options: Compactness weight *	Defines the compactness factor, in range for the oversegmentation's step.
DepthSynth options: Samples *	Indicates how many additional depth samples will be found in depth maps images.

(*) Advanced parameter: please do not use unless you know what you are doing.

How to use CR-PLAY mixed pipeline in Unity

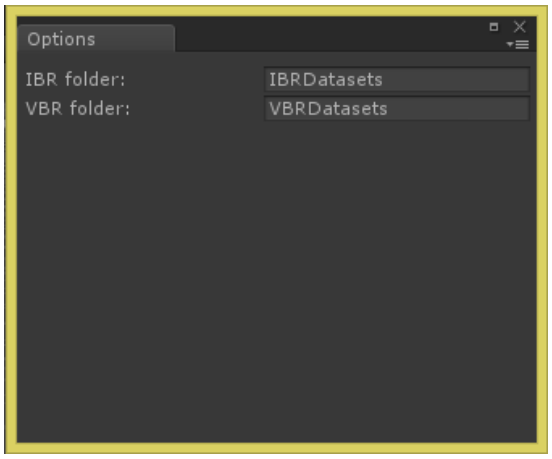
Installation

To install the CR-PLAY Mixed Pipeline in Unity, it is necessary to import the unity package **`cr-play.unitypackage`**, that automatically installs all needed resources in the selected Unity project. This package contains both IBR and VBR scripts and resources.

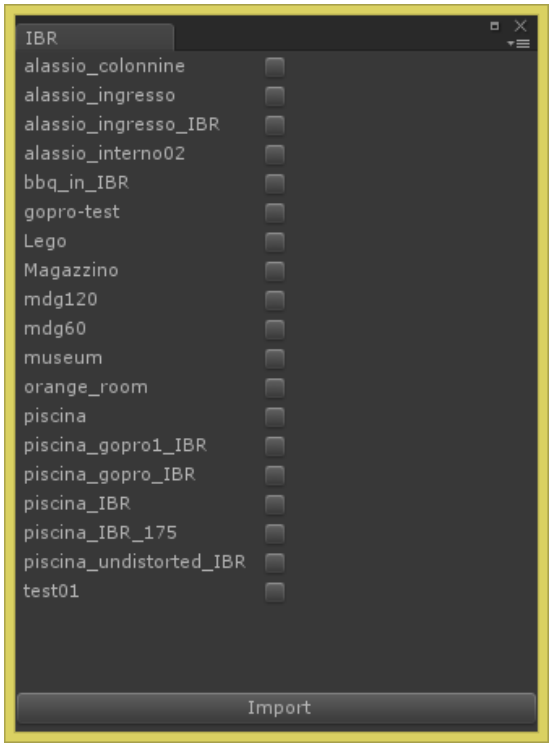
Import IBR(VBR) datasets

Before working with IBR(VBR), datasets must be imported to the Unity project using the dedicated tools provided. Importing operations automatically take the dataset zip coming from the Reconstruction Tool and import the needed assets in the Unity project folders.

- Copy and paste the dataset file coming from the reconstruction tool in the **Assets/IBRDatasets(VBRDatasets)** folder (create this folder if needed)
  [Default datasets folder can be changed by selecting the **CR-PLAY/Options** menu.]



- Select the **CR-PLAY/Import IBR(Import VBR)** menu to open the Import window.
- Check the dataset you want to import and click **Import** to import the IBR(VBR) datasets.

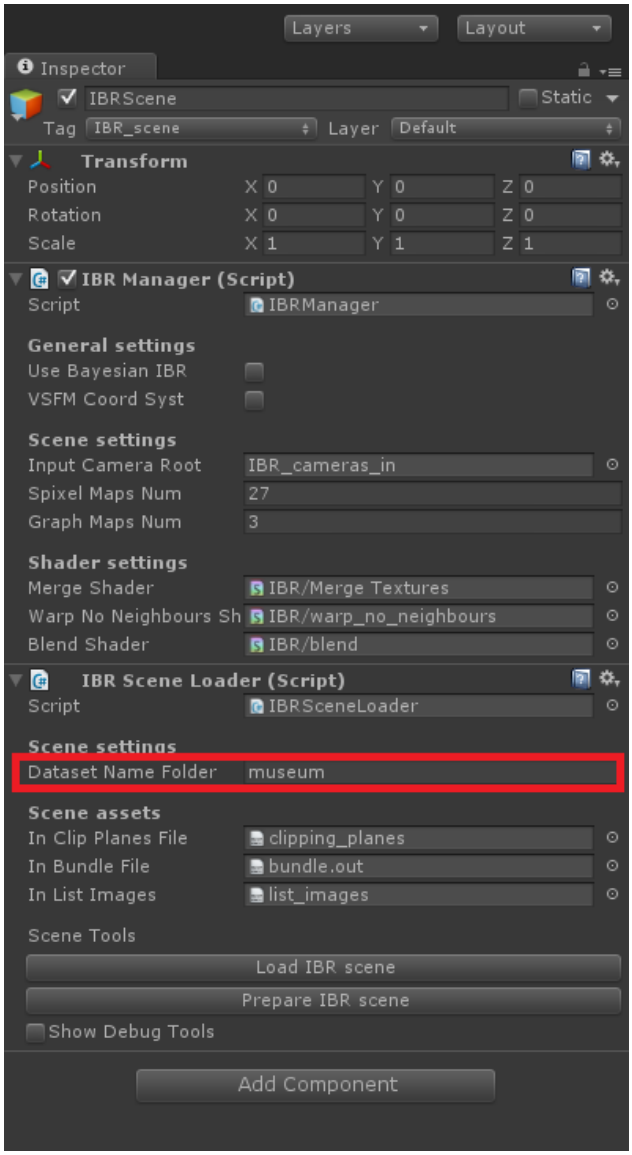


IBR(VBR) datasets are automatically imported in **Assets/StreamingAssets/<dataset_name>** folder and **Assets/IBRAssets(VBRAssets)/<datase_name>**, depending the asset nature.
[StreamingAssets is a special Unity folder that is automatically copied in the deployment packet, IBR asseets need to be copied in the deployment packet because IBR C++ Plugin needs to use them during rendering]

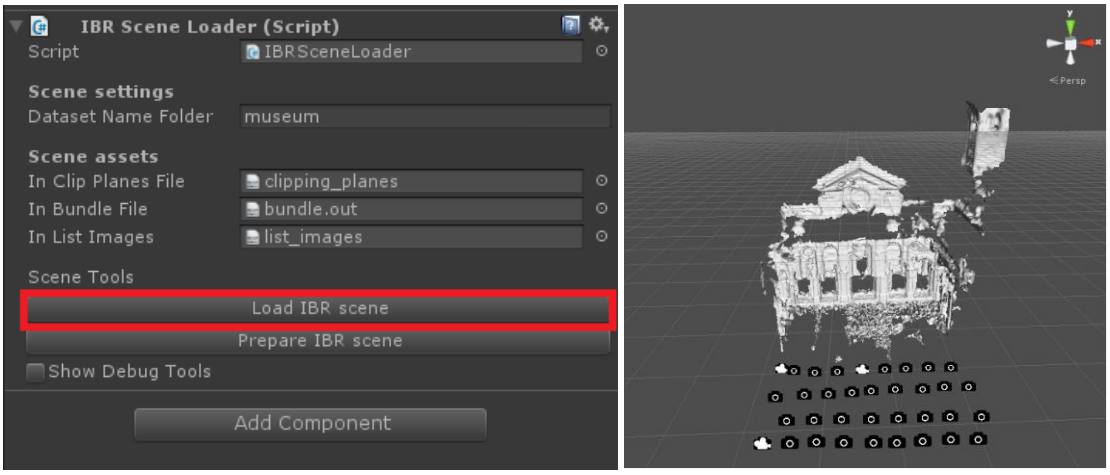
Setup IBR scene

To setup the IBR scene it is necessary to load the pre-configured IBRScene prefab (**Assets/Prefabs/**) in the current scene and initialize the associated behaviors to let the IBR scene to load a particular dataset.

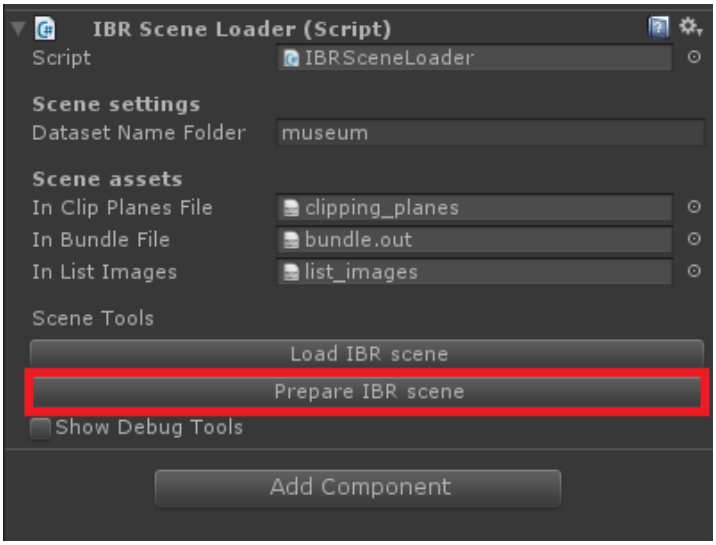
- Drag and drop the **IBRScene** prefab in the scene view (or in hierarchy view).
- Attach the **IBRRender** behavior to the Main Camera by drag and drop the IBRRenderer.cs script on the main camera game object.
- Select the **IBRScene** game object in the Hierarchy View to open its Inspector View.
- Go to the **IBRSceneLoader** behaviour and set the "Dataset Name Folder" field with the <dataset_name> to be loaded.



- To load the IBR scene in Unity, click on the "Load IBR scene" button in the **IBRSceneLoader** behaviour.



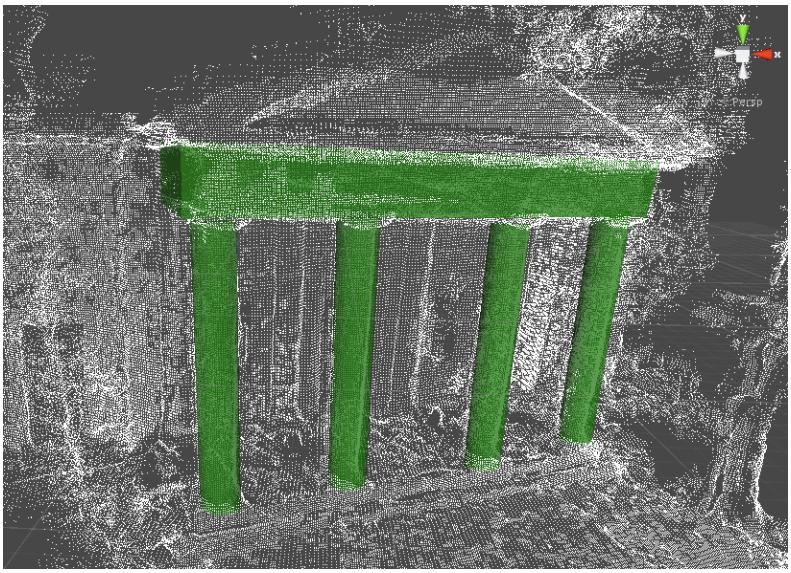
- To generate the superpixel textures and all the needed data, click on the "Prepare IBR scene" button in the **IBRSceneLoader** behaviour; this step may take some time depending on the number of input images. [This operation need to be performed only the first time a dataset is loaded or every time a dataset is changed].



Use of proxies for Physics Collisions and Depth Occlusion on IBR scene

Thanks to the point cloud generated by the Reconstruction Tool, it is possible to have an overall representation of the theoretical virtual spatial occupation of the IBR scene and use 3D models as proxies to simulate specific behavior, such as physics collision and depth occlusion.

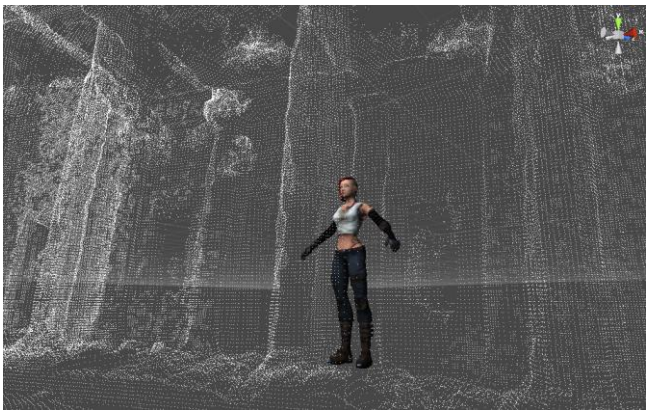
Physics Collisions can be simulated by placing an invisible collision mesh where the point cloud indicates that a specific portion of the IBR scene will be rendered. The image below show an example of collision mesh in green.

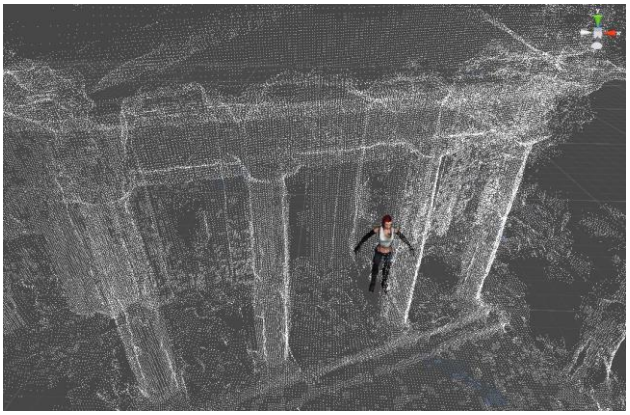


To do this the user has to place the appropriate mesh in the scene, add a **MeshCollider** component and set an invisible material to its **MeshRenderer** behaviour.

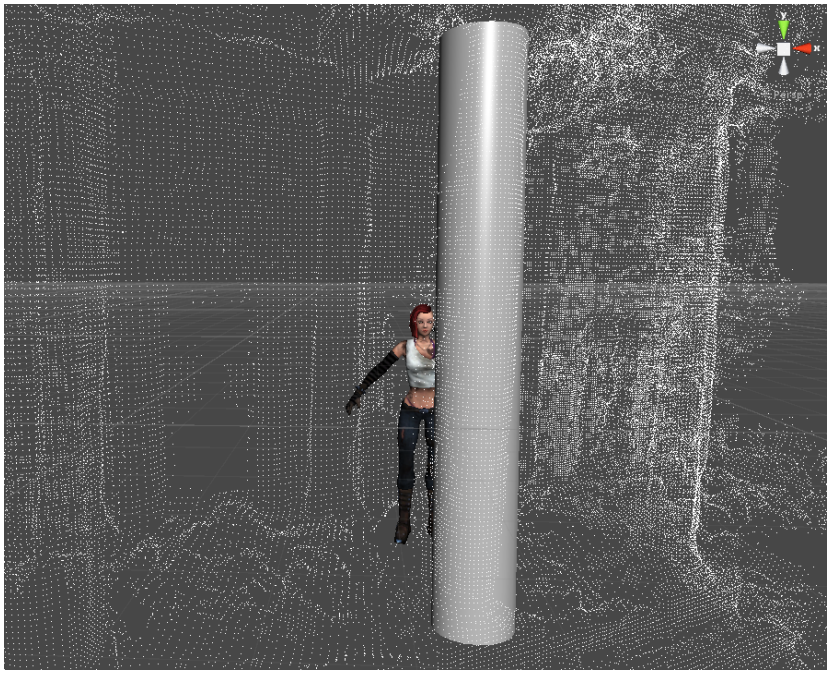
Depth Occlusion simulates depth information in an IBR scene by using 3D depth proxies placed in the 3D world using the spatial reference provided by the point cloud.

The procedure to set a depth occlusion proxy is more complex, so let us consider the example of a 3D character model placed behind the column of a small temple rendered using IBR.

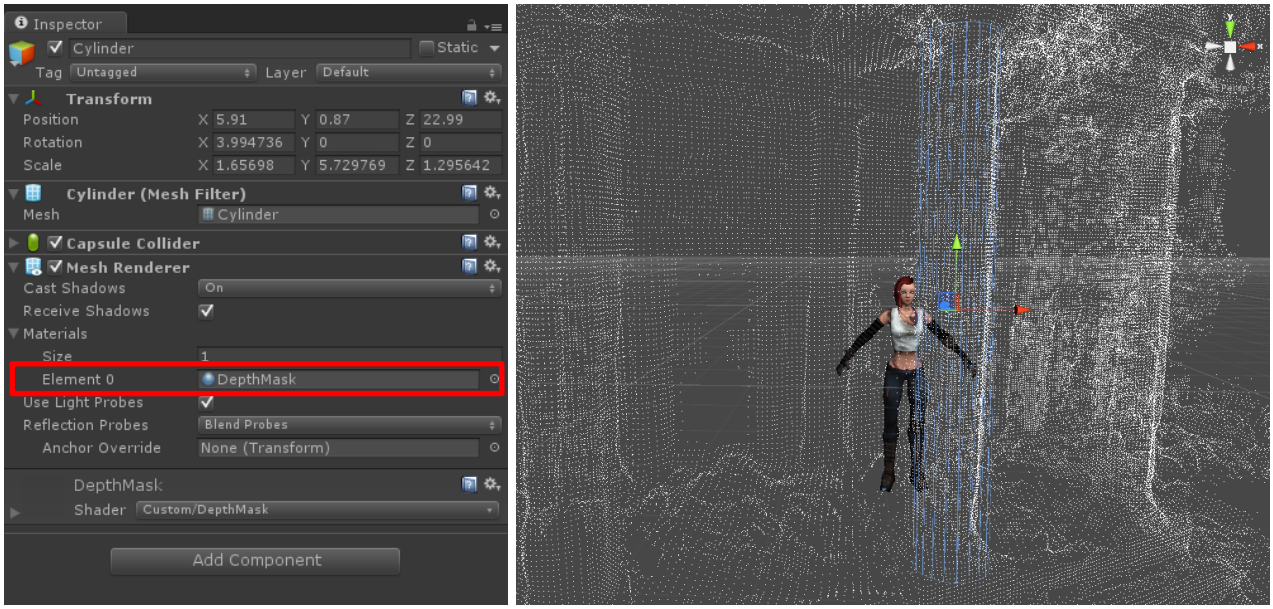




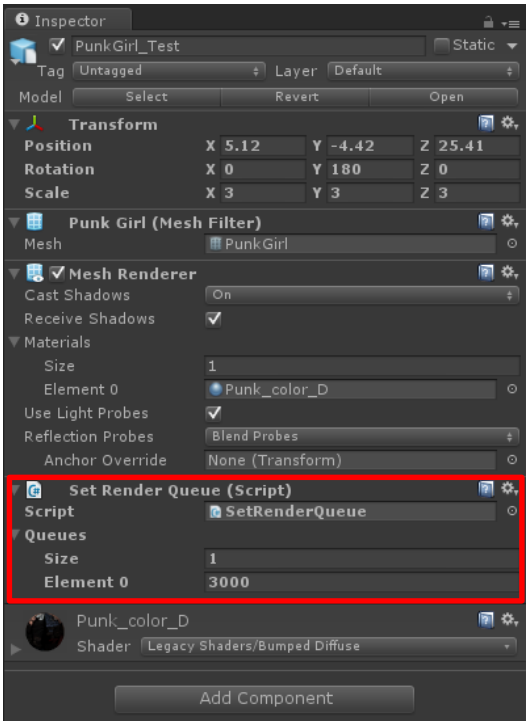
In order to simulate the IBR depth information, a **3D proxy cylinder** is placed where the column will be rendered (using the point cloud as a spatial reference).



The next step is to ignore the geometric information of the cylinder by applying the **DepthMask** material that can be found in folder **Assets/Materials**.



Additionally to the User must change the Render Queue position of the 3D model that needs to be masked. This operation can be performed by adding the script behavior **SetRenderQueue** to the target model. The script can be found in the **Assets/Scripts/Utils** folder.



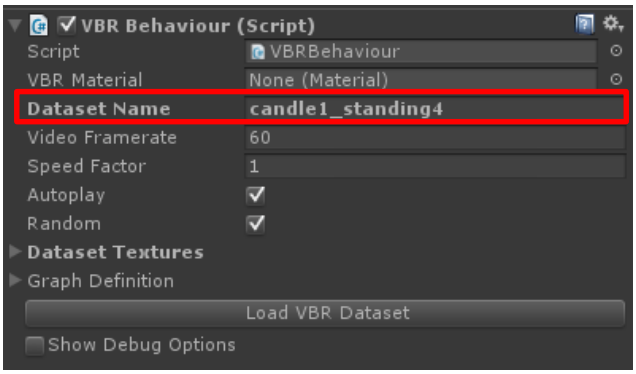
As a result, the proxy model pulls the background (the IBR scene) in front of the 3D target model according to the depth information provided by the proxy.

The same 3D proxy model can be used for both the Physics Collision and Depth Occlusion purposes.

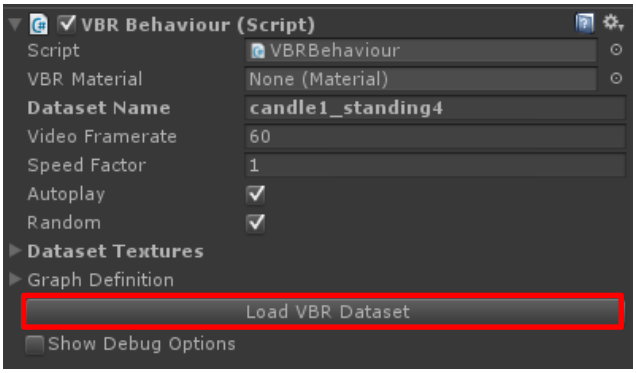
Setup of the VBR scene

To setup the VBR scene it is necessary to load the pre-configured **VBRBehaviour** prefab **Assets/Prefabs/** in the current scene and initialize the associated behaviors to let the IBR scene to load a particular dataset.

- Drag and drop the **VBRObject** prefab in the scene view (or in hierarchy view).
- Select the **VBRObject/VideoTexture** game object in the Hierarchy View to open its Inspector View.



- Go to the **VBRBehaviour** behaviour and set the "Dataset Name" field with the <dataset_name> to load and the "Video Framerate" field with the frame-rate of videotextures.
- To load videotextures, click on the "Load Dataset" button in the **VBRBehaviour** behaviour.



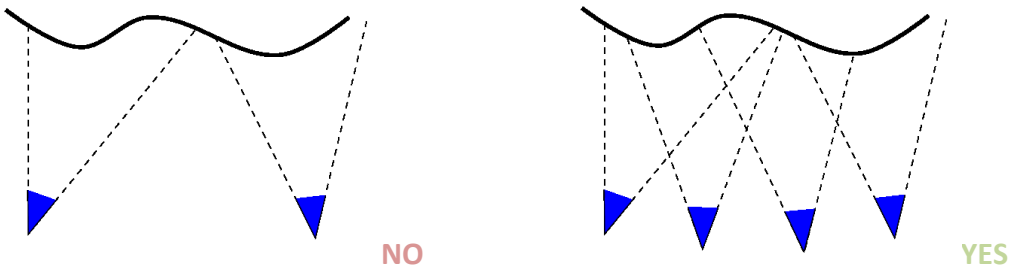
Guidelines for capturing datasets

Camera settings

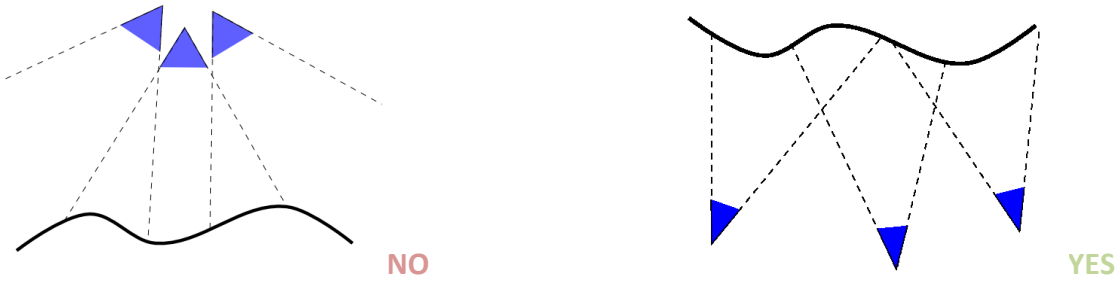
- If your camera has a resolution setting, choose the highest resolution possible. If needed, the algorithms will downscale the images later.
- Set a fixed aperture, either using aperture priority or manual mode (consult the camera manual if necessary).
- Take images that are well-exposed (neither too dark nor too bright). This might require you to change the exposure time or ISO speed if you are in manual mode. In aperture priority mode, check that the camera selected sensible values for these settings (consult the camera manual if necessary). Make sure that the exposure time DOES NOT CHANGE from one picture to another.
- If the depth of field is too small, try setting a larger f-Number (smaller aperture) but be aware that this might darken the image if the exposure time is not increased accordingly.
- Focus! If the scene or object are out of focus, reconstruction will fail. It is ok to refocus in each of the images.

Camera placement

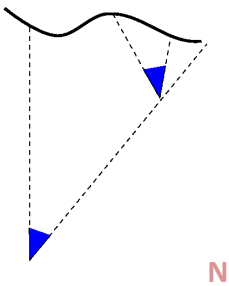
- Take pictures with sufficient overlap. If in doubt, just capture images at small intervals. This takes more time but ensures that everything has been seen with sufficient overlap. Each point on the surface should be visible in at least four images.



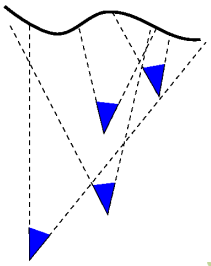
- Do not take panoramas. The reconstruction will not work if you just rotate the camera while standing still. The algorithms need to observe the surface from different viewpoints. The further away from the object, the larger the translation between shots can be.



- It makes sense to take “connecting shots”. For example if you take images from farther away and then move closer to capture some details, the algorithm might not be able to connect the details to the large scale surface even though they overlap. Try to take some additional shots while moving towards the details.



NO



YES

- It is allowed (and can actually be quite beneficial) to take images from different heights. So, if you have the chance to climb on a nearby wall, do not hesitate to do so.

Scene content

- The reconstruction is based on the assumption of a static scene. Try to avoid moving parts such as leaves, shadows, or pedestrians as much as possible. A certain amount of movement is tolerable but those parts will be reconstructed less accurately or not at all.
- Surfaces with texture variation work well. If your object does not display a lot of texture, it can help if at least some other textured surfaces are visible in the images. In the case of blank walls, it will help if you add a poster for example.
- Transparent and mirroring objects do not work. It is possible to have a car in a scene as long as the light is not too bright (for example on a cloudy day or in shadow) and the car does not cover a large part of the image.
-

Best practices (if possible) and additional information

- Try to do (at least) two (semi-) circles with different radii around the object. Don't forget to add transitions pictures if you want a smoother experience in specific regions.
- Surface points that are observed under grazing angles in all images are hard to reconstruct.
- Reconstruction might work better on overcast days (even illumination, no moving shadows, …).
- Take overview images as well as close-ups for specific details (but remember the connecting shots).

Annex C – Gameplay examples

Fighting Games

The first example for usage is a fighting game with left-right movements. The game camera can move left-right, but also rotate a bit around Y-axis. If more rows of cameras are captured, zoom in-out can be done as well.

Figure 21 - (example from Mortal Kombat X): fighters are facing each other moving from left to right and vice-versa. Camera does the same movement, keeping them in the centre of the screen.

Figure 22 - (example from Mortal Kombat X): while doing special moves, camera rotates around Y-axis to emphasize the particular moment of gameplay.

For such a scenario, the capture of environment should be done by moving on a line, taking pictures from left to right (or vice-versa), with sufficient overlap of the environment between two consecutive photos.

In order to obtain the rotation of the camera as in Figure 22, the same line of capture should be done, but keeping the camera rotated (as it is intended to be in the game).

Doing multiple lines at different distances from the environment allows for a zoom in-out effect in the game.

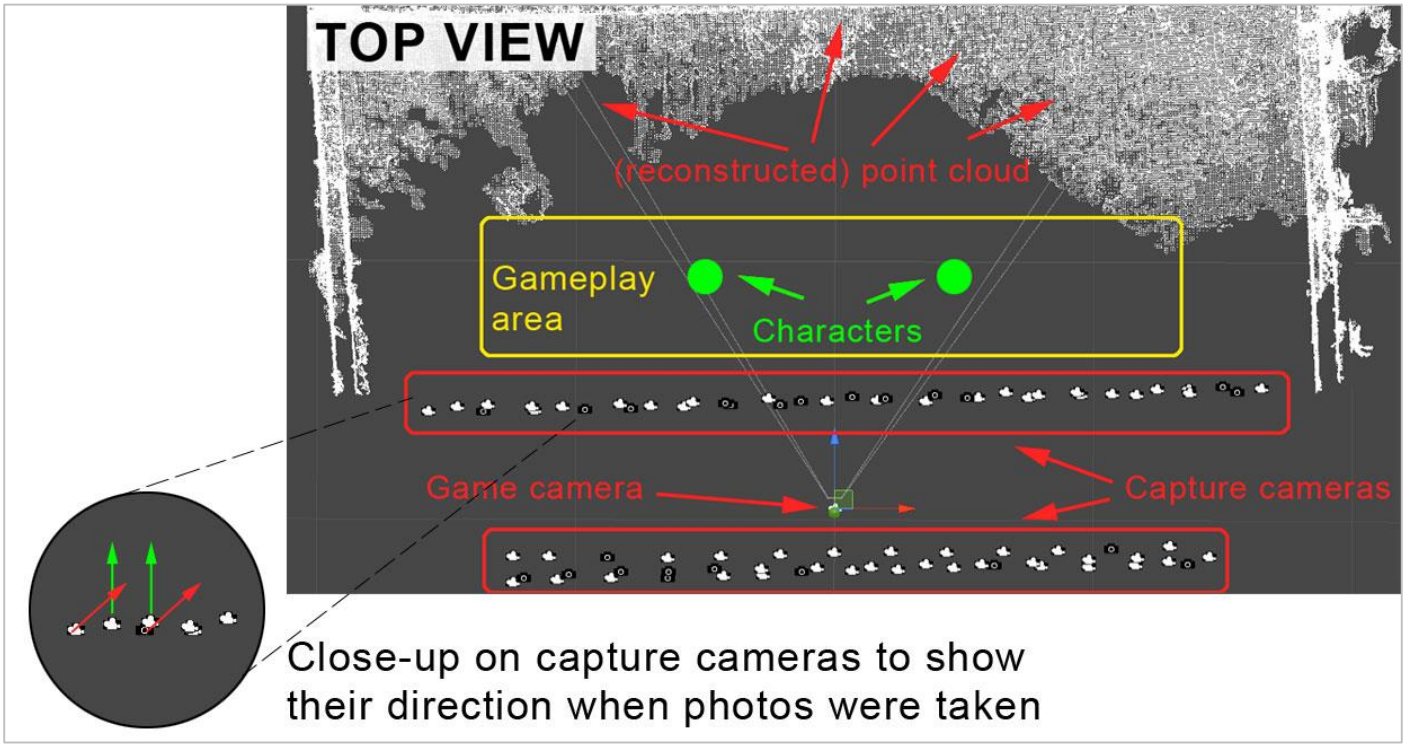


Figure 23 – A snapshot from Unity showing a scene being prepared for a fighting game. Photos were taken from two parallel lines, with cameras facing forward and slightly left-right.

2D (2,5D) Platformers

The second example of games is for 2D platformer games. The idea is to capture a scene from a frontal point of view, and create the gameplay taking advantage of the captured area. Thus platforms, obstacles etc. will be created in 3D and integrated with the captured scene. Imagine a Rayman level (where movement is possible in four directions left-right-up-down) where the environment is done with CR-PLAY technology.

Camera can also rotate a bit to obtain the 2,5D effect.

Figure 24 - mockup inspired by RAYMAN

In this type of game, character can jump, run, hit objects etc. by using 3D objects added on scene by game dev. The captured environment is bigger than just one screen, so the camera will pan to keep the character in the centre of the screen, maximizing the 3D effect.

Capture of the environment should be done by covering the entire game area, keeping in mind the type of different cameras that are needed in the game (i.e. just frontal or also slightly rotated).

Angry Birds-like games

Similar to 2D platformers, with a different gameplay.

Figure 25 - snapshot from Angry Birds 2

Imagine substituting the background in Figure 25 with a scene captured from a real environment. Of course platforms, birds, pigs will be created using traditional 3D assets.

Camera will pan in four directions, emphasizing the 3D feeling of the scene.

FPS

Gameplay is a FPS, where players can rotate limited to certain degrees. The movement of the character is also restricted to avoid having the camera go outside the capture field.

Figure 26 - snapshot from Silver Arrow, game prototype made with low-fidelity prototype of CR-PLAY Technology – Video is available at https://youtu.be/mKXsoB3OpKU

Capture cameras should be placed accordingly with the intended rotation of game camera and space of movement of the character.

Railway shooters

Similar to FPS, but camera movement is controlled by the game, typically along a rail. The player moves the crosshair with the mouse to hit targets.

Figure 27 - snapshot from Rage HD

The design of the rail along which the camera moves should be accurately thought out in advance and used as a reference when capturing the environment.

Point&Click Adventures

The player interacts with the mouse on objects/parts of the screen that can be either 3D or captured. The captured scene allows for camera movements managed by the game (i.e. controlled pan/rotate).

Figure 28 - snapshot from Syberia II

Multidirectional shooters

Imagine a far-west scenario where outlaws appear from windows, behind a corner etc. Players can move the camera left-right-up-down, and control the crosshair with the mouse to shoot them down.

Figure 29 - mockup image showing a possible scenario for multidirectional shooter

Urban sports

Games such as Basketball and Squash can be done in a captured urban scenario, with some 3D elements like the basket, player etc. The camera can move to some extent.

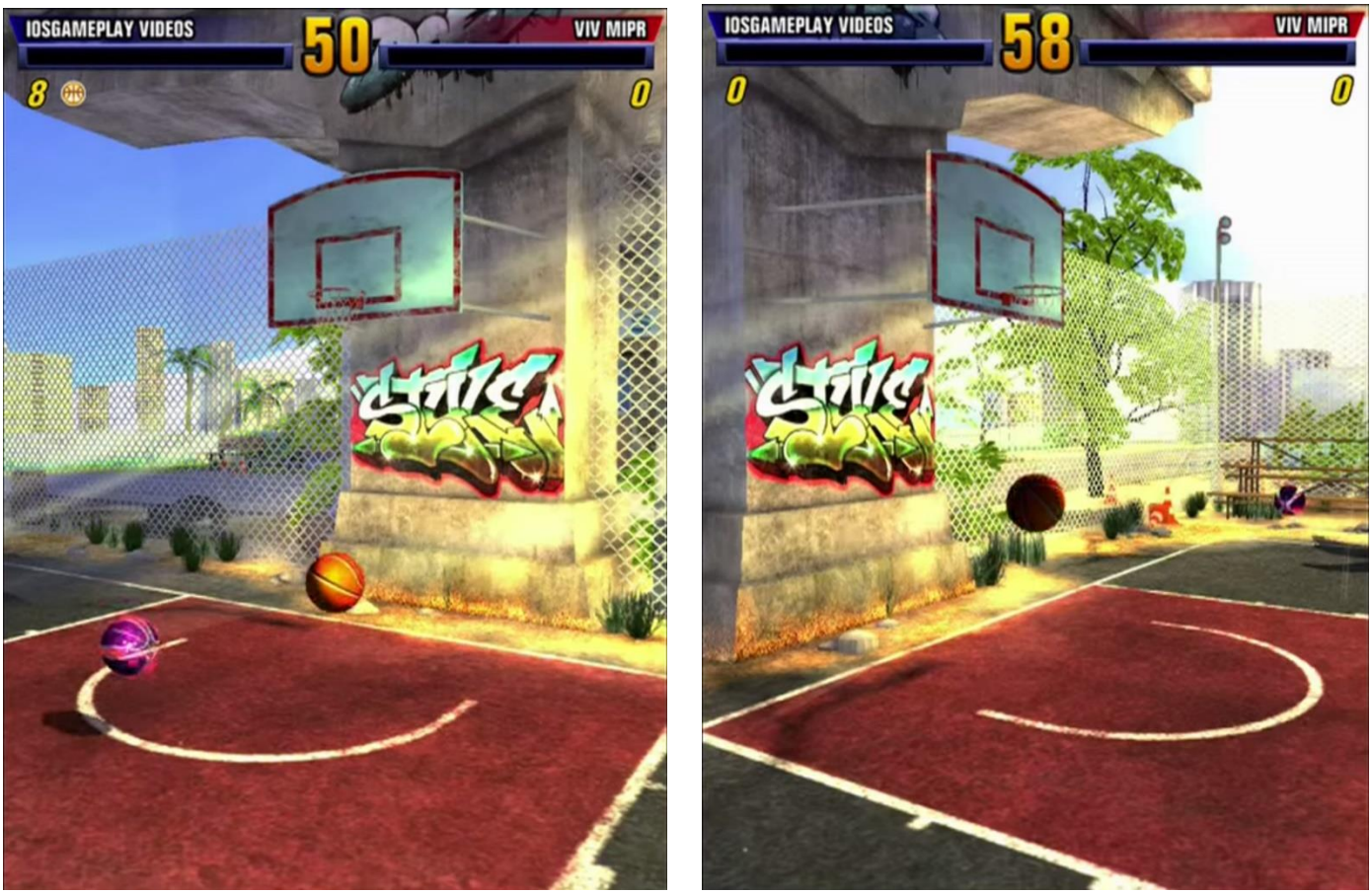


Figure 30 - snapshots from SuperHoops, iOS game