OptiSpace: Automated Placement of Interactive 3D Projection Mapping Content

Andreas Fender[1], **Philipp Herholz**[2], **Marc Alexa**[2], **Jörg Müller**[3]

[1]Aarhus University, Denmark; [2]TU Berlin, Germany; [3]University of Bayreuth, Germany

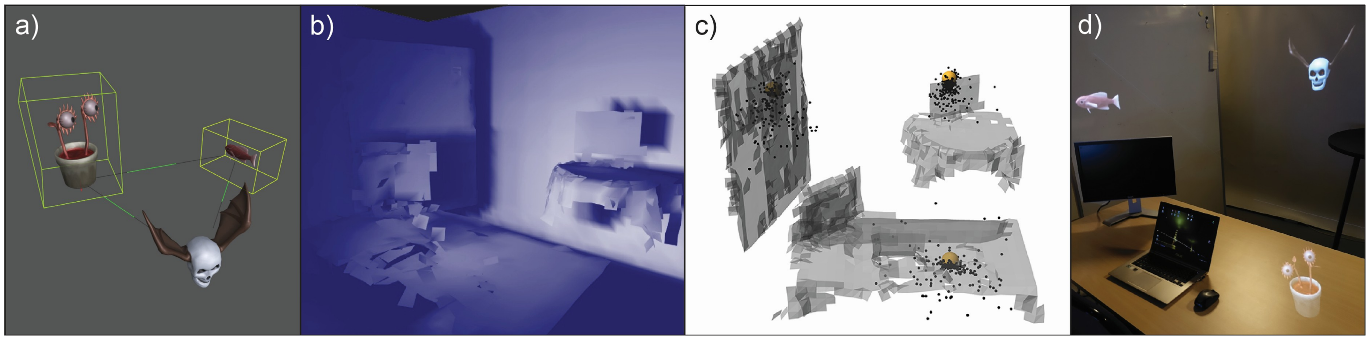


Figure 1. OptiSpace enables the development of interactive room-scale 3D projection mapping applications independently of the target environment. (a) Developers create virtual scenes and define spatial attributes. (b) OptiSpace measures room geometry and user viewing behavior in the target environment, including surface visibility. (c) OptiSpace optimizes the placement and projection of virtual objects in the target environment. We use an extended version of covariance matrix adaption. Optimal positions are depicted in yellow. (d) The content is placed such that it can be projected, is visible to the user from as many viewpoints as possible, and satisfies additional constraints specified by the developer.

Abstract

We present OptiSpace, a system for the automated placement of perspectively corrected projection mapping content. We analyze the geometry of physical surfaces and the viewing behavior of users over time using depth cameras. Our system measures user view behavior and simulates a virtual projection mapping scene users would see if content were placed in a particular way. OptiSpace evaluates the simulated scene according to perceptual criteria, including visibility and visual quality of virtual content. Finally, based on these evaluations, it optimizes content placement, using a two-phase procedure involving adaptive sampling and the covariance matrix adaptation algorithm. With our proposed architecture, projection mapping applications are developed without any knowledge of the physical layouts of the target environments. Applications can be deployed in different uncontrolled environments, such as living rooms and office spaces.

INTRODUCTION

3D projection mapping creates the illusion of volumetric virtual objects by projecting content such as 3D models onto physical surfaces with real-time perspective correction. This technique is promising in terms of providing augmented reality for uninstrumented users [6].

CHI 2018, April 21–26, 2018, Montreal, QC, Canada

© 2018 ACM. ISBN 978-1-4503-5620-6/18/04…$15.00

DOI: https://doi.org/10.1145/3173574.3173843

One core difficulty is that not all surfaces are suitable for projection mapping. Surfaces might be out of the field-of-view of the projectors. Furthermore, many dark or reflective surfaces cannot be projected on, despite advances in projection technology and projector calibration. Large distances between virtual objects and physical surface or virtual objects that spread over multiple physical surfaces with different distances from the user, cause contradicting depth cues (see Figure 2). A mismatch in motion parallax of real and virtual edges makes the object appear as if it were moving when viewing it from different angles. Furthermore, perspective correction potentially requires a lot of physical space (see Figure 3). Consequently, the placement of virtual content within 3D projection mapping scenes is very difficult.

The most widely used approach for addressing this problem is a careful manual design of content placement and/or projection surfaces. However, if projection mapping is to be used in common living rooms [17] and offices [20], this approach clearly is not practical, since it requires an expert to be present in the target environment.

Current solutions to this problem entail either manual input from end-users [15] or from content designers [19]. Other approaches are limited to 2D [21, 9, 22] or ignore user perspectives [25].

To solve the problem of automatic content placement for 3D projection mapping, we present OptiSpace, which:

1. analyses the geometry of physical surfaces and the viewing behavior of users over time using rgbd cameras,
2. clusters measured user perspectives,

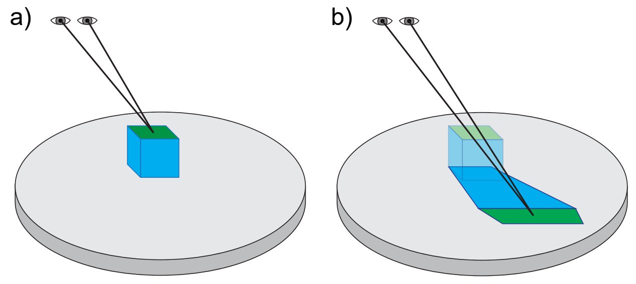


Figure 2. A user is watching a virtual cube, which is projected onto a physical surface using perspective correction. The eyes are not focusing on the intended virtual surface (a), but on the physical surface behind it (b). Bipolar vision, eye convergence and eye accommodation provide depth cues, which eliminate the illusion of a volumetric virtual object. However, with increasing absolute distance between the user and the virtual object, the problem becomes less apparent.

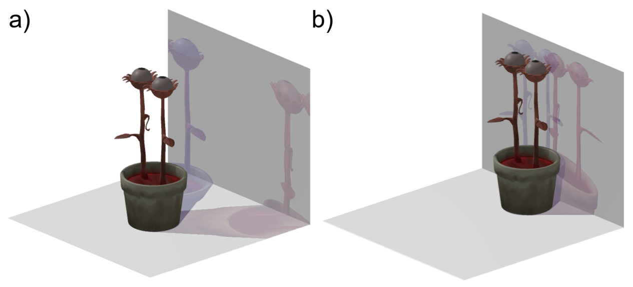


Figure 3. Virtual objects require nearby physical surfaces for projection mapping. (a) The virtual object (opaque) is far from the physical surface. Perspective correction requires a lot of physical space when viewed from the right (blue shade) or left (red shade). It extends beyond the available surface. (b) The virtual object is closer to the physical surface and perspective correction requires less space. Furthermore, the negative effects of contradicting depth cues are decreased.

3. simulates the scene users would see if content were placed in a particular way,
4. evaluates the simulated scene according to perceptual criteria, including visibility and the visual quality of virtual content,
5. optimizes content placement, using a two-phase procedure involving adaptive sampling and the covariance matrix adaptation algorithm.

Figure 1 provides an overview. With our architecture, projection mapping applications can be developed once, without any knowledge of the physical layout of the target environment. This is relevant, because a lack of scalability of projection mapping content is a major factor constraining spatially augmented reality in uncontrolled environments such as living rooms.

We envision many different room-scale projection mapping applications ranging from games and entertainment to 3D telepresence in the living room. For example, in a projection mapping telepresence application, multiple participants could be scanned in 3D at their respective locations. Their scaled-down bodies could be shown as projection mapped models in each others' rooms, where they could also move and walk around. The system places each participant optimally in each room. The local user can walk around, while all participants are always shown correctly.

RELATED WORK

Projection mapping

The creation of augmented reality for uninstrumented users through placement of projectors and cameras in interactive rooms is a vision, which has been around for a long time [20]. Today, this vision can be realized simply with off-the-shelf components [28], which has spawned an active research field of interactive projection mapping. Content can be 2D, and appear to be on physical surfaces in the room. Alternatively, the user's perspective can be tracked, and perspective correction can be applied, such that it appears to be placed in 3D space inside the room. A large variety of systems demonstrate the potential and benefits of this approach.

To name just a few examples, *RoomAlive* [17] provides a calibration approach for multiple projectors and cameras, as well as a number of interactive games and experiences. *Room2Room* [19] presents a telepresence application in projection mapping augmented reality. *The Other Resident* [1] provides an artistic experience using interactive projection mapping.

Content placement

One core difficulty of interactive projection mapping is the placement of content, which needs to be adapted to the actual room geometry of each target environment. We focus on uncontrolled target environments like living rooms, i. e., the geometry is unknown during development.

One approach to addressing this issue is *simplified manual content placement by the user*. For example, with *Ubi Displays* [15], users define display and touch regions in the environment manually. The main drawback of this approach is that users might find manual content placement cumbersome and possibly difficult.

With *semi-automatic content placement*, the content creator manually defines a set of possible content areas, while the system automatically decides between them. This approach is followed in the *Room2Room* system of Pejsa et al. [19], a conferencing system for the living room, which displays the remote participant using 3D projection mapping. The system takes missing depth cues into consideration, as well as other constraints like preserving distance between participants.

Fully automatic content placement systems place content without any intervention from content designer or user. This approach is related to the problem of *view management* for head-mounted AR systems [2]. With respect to this problem, AR content (e.g., labels) is placed within the field of view of a user, so as to maximize legibility. Most automatic content placement approaches for projections reduce the problem to 2D, and many ignore user perspectives. *SurroundWeb* by Vilk et al. [25] displays web content around a TV in a living room. Their solution is an abstraction for rendering onto flat surfaces, without disclosing the actual room layout to third party applications. [21, 9, 7] automatically identify free regions with good reflectance properties for projection. [22] takes occlusion from the user's point of view into account.

One drawback of placing content according to the instantaneous user perspectives is that placement becomes sub-optimal as soon as users move. In such cases, content needs to be moved, which might not be desirable in the view of layout consistency and spatial memory of the user. This issue can be addressed by aggregating user perspectives over time and *placing content according to aggregated user perspectives*. In our prior work [8], we addressed the problem of placing physical displays in a multi-display environment. We tracked user perspectives with Kinect cameras and created volumetric heat maps of user attention in the space. Based on these heat maps, we optimized display positions on physical surfaces using joint gradient descent. The main difference to our current paper is that we address the more general, and indeed more complex problem of placing content in 3D. Further, in addition to optimizing over volumetric heat maps, we render the view from user perspectives, leading to a more flexible solution.

Projector calibration
An alternative way to deal with surfaces that are difficult to project on is projector calibration [5, 4]. High-precision calibration techniques compensate for reflectance properties by using radiometric compensation (see for example [26], [10]). These techniques allow for a wide range of possible surfaces to be used for projection in non-optimized environments. Some of the approaches use multiple and largely overlapping projectors [24, 18]. However, uncontrolled environments generally contain objects and surfaces, which exceed the hardware or physical limitations of projectors, for example, in terms of luminance, or are shaped so that they cannot be illuminated entirely, because of self-shadows. The properties of these surfaces cannot be compensated for, even with previously discussed techniques. Furthermore, calibration techniques generally only account for the physical properties of the environment, but not for how it is used and with regard to which surfaces are generally well visible for users. Projector calibration is therefore complementary to our approach.

In summary, our main contribution, compared to related work, is the *fully automatic content placement in 3D* based on *aggregated user perspectives*. This has the core benefits of scalability through independence of content from room geometry, as well as robustness for moving users.

SYSTEM OVERVIEW
This section provides a brief overview of the main components and our example setup.

Example setup
Our example setup consists of three Kinect depth cameras and three *BenQ* full HD projectors, as can be seen in Figure 4. Projector and Kinect postures are calibrated using the *RoomAlive Toolkit* [17]. All examples in this paper use this arrangement of Kinect and projector devices, but various differing room layouts.

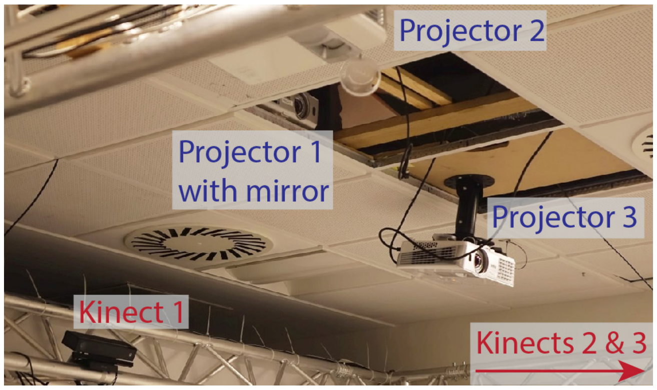
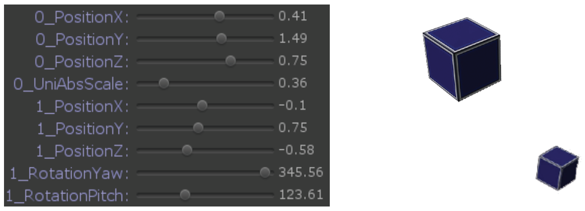


Figure 4. Projector and Kinect arrangement of all the examples given in this paper. Projector 1 is illuminating downwards using a mirror, typically for projecting onto a table. Projectors 2 and 3 are illuminating the room's walls or large objects like whiteboards. The three Kinect devices are capturing the environment from three orthogonal viewing angles. Kinects 2 and 3 are not in the picture.



```
0_PositionX:    0.41
0_PositionY:    1.49
0_PositionZ:    0.75
0_UniAbsScale:  0.36
1_PositionX:    -0.1
1_PositionY:    0.75
1_PositionZ:    -0.58
1_RotationYaw:  345.56
1_RotationPitch: 123.61
```

Figure 5. A design vector for a scene with two cubes. The first cube can be positioned and scaled uniformly. The second cube can be positioned and rotated around the X and Y axis. This defines the design space of this scene for the optimization algorithm.

OptiSpace components
OptiSpace consists of three phases:

In the **design phase**, the developer creates the virtual scene and mainly specifies parameters to control the placement. There is no knowledge about the target environment at this point. We discuss the details in the *Design phase* section.

When deploying in a target environment, the **data acquisition phase** is executed. The visibility and suitability of surfaces in the target environment are measured while users interact within it. Furthermore, static surfaces are reconstructed and user viewpoints are measured and sampled. This phase is executed once per target environment, before the actual application is running. We implemented a dedicated *OptiSpace Data Acquisition Application*, which generates the output data. We discuss the details in the *Data acquisition* section.

During the **optimization phase**, virtual content is placed based on the acquired data and on object-specific properties defined by developers. The optimization phase has to be executed once per application per target environment after data acquisition. We discuss our optimization in the *Optimization* section.

DESIGN PHASE
Developers of an OptiSpace application need to define certain parameters to be later used during optimization.

Design vector v_d

The design vector describes the aspects of the scene that the content developer wishes to adjust to the room geometry. It is given by $v_d \in \mathbb{R}^{N_d}$, where N_d is the number of entries of the design vector. The dimension of the design vector and the semantics of the entries depend on the scene to be optimized. In the simplest case, only the positions of objects are adjusted, as described by three design vector entries per object. Developers can combine different semantics and set min and max values to build the design vector. For instance, we also implemented an automatic adjustment of the scale and rotation of virtual objects. Scale can be uniform or anisotrop, absolute or perceived, i.e., relative to viewpoint distance. Rotation can be a combination of yaw, pitch and roll. The optimization algorithm uses the defined design vector to control the scene state. Defining the design vector elements is crucial for achieving the desired optimization results.

Layout quality Q_L

Developers can improve the optimization results by providing a layout quality function Q_L to indicate desired spatial attributes of virtual objects and relationships between virtual objects. This is mostly used to implement soft constraints. Content developers can specify preferred sizes of objects, minimum distances between objects, or other criteria. The output of the function has to be a real number between 0 (lowest quality) and 1 (highest quality). For example, for the scene in Figure 5, developers can define a preferred scale s_p for the first cube and a minimum distance d_p between the cubes. The layout quality function is then given by:

$$\begin{aligned} Q_L &= q_s \cdot q_d \\ q_s &= \frac{1}{1+||s-s_p||^{\phi_s}} \\ q_d &= 1-\frac{1}{1+max(0,||\vec{p}_1-\vec{p}_2||-d_p+1)^{\phi_d}} \end{aligned}$$

Here, s is the design vector element containing the current uniform scale for the first cube. $\vec{p}_1$ and $\vec{p}_2$ are the positions of the cubes. ϕ_s is an exponent for defining how strictly the size has to be at the preferred level. ϕ_d defines how sharply the quality declines when objects have a distance around d_p.

Developers are free to define any terms and functions in order to calculate the layout quality.

DATA ACQUISITION

When deploying an OptiSpace application, the environment and the users are tracked for a certain period of time to *measure* the suitability of surfaces for projection. The data is then *post-processed* to generate output data to be used by the optimizer and the target application.

Before starting the data acquisition, installers have to set up projectors and depth camera devices in the target environment. All projector and depth camera postures have to be known. For instance, OptiSpace can use the output intrinsics and extrinsics of a *RoomAlive* calibration [17].

Measurement

After calibration, the measurement can be started (see Figure 6.1). Users interact in the environment as they would normally (see Figure 6.a).

Capture

Based on the real-time streams of the Kinect devices, a point cloud and triangulation of the current room geometry is created at every frame. For every surface point, the local surface brightness is measured, using the Kinect infrared streams. The skeleton tracking functionality of the Kinect SDK provides information about users' head positions and orientations. We merge multiple skeleton streams into one global representation similar to [23]. Figure 6.b is an example of a frame during capturing.

Voxel grid

During measurement, the environment is split into a regular grid of voxels (see Figure 6.c). Each voxel contains the following data, which is updated with every frame during capturing:

- **Geometric persistence**. Measures how persistently surface points were present within the voxel (similar to [8]).
- **Illumination voxels**. Measures whether the voxel is illuminated by a projector and how bright the surface is at that voxel. Illumination is only measured whenever the voxel contains surface points during that frame. We check whether the voxel is within a projector's view frustum and perform raycasts from the projectors to the voxel. If the ray hits the mesh of the real-time capture before reaching the voxel, then the voxel is in a shadow and we set the illumination to zero. If the line of sight is free, i.e., the voxel is illuminated, we determine the surface brightness from the infrared streams. If the surface brightness is below a certain threshold, i.e., a dark surface does not reflect much of the projector light, then we set the illumination to zero.
- **Visibility voxels**. Measures how visible the voxel is for users (similar to [8]). This takes into account how often the voxel gets occluded, the average viewing angle and whether the voxel is generally in the central field of view of users. Visibility is only measured whenever the voxel contains surface points during that frame.
- **View voxels**. Measures, how long a user's head was within the voxel.

All the calculations are made every frame and the results are normalized at the end.

Post-processing

The output data is based on the voxel grid. First, we reconstruct the static parts of the environment, based on voxels with high persistence (see Figure 6.2). The illumination and visibility values from the voxel grid are transferred to the UV coordinates of the reconstructed mesh to allow for fast GPU access and interpolation. The resulting data on the mesh after the post-processing is visualized in Figure 6.2. Figure 6.f shows the final result, combining illumination, visibility and viewpoint clusters. The raw voxel grid is no longer needed after post-processing is complete.

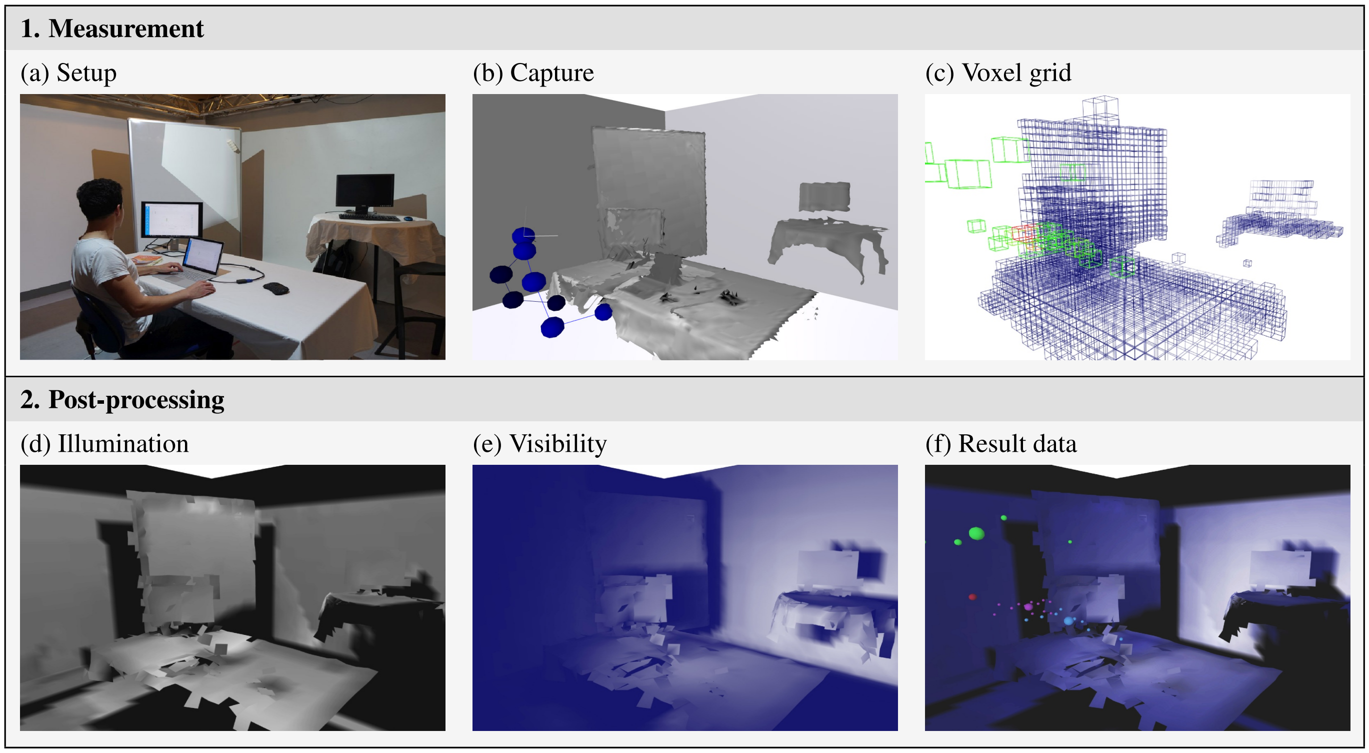


Figure 6. The data acquisition is split into (1) measurement and (2) post-processing. (a) The user interacts in the environment. For this photo, projectors emit white light so as to highlight projector shadows. (b) The surface geometry and the user are captured in real-time. (c) A voxel grid stores surface data (blue) and viewpoints (green to red). The user mostly looked from the viewpoint voxel in red and occasionally moved through the green ones. (d) Illumination of the static surfaces by the projectors over time. (e) Surface visibility over time (white: well visible). (f) Combined data. The view voxel centers (small spheres) are clustered to create representative viewpoints (big spheres).

Clustered viewpoints $\vec{u}_i$

We cluster the raw viewpoint voxels into a set of N_v viewpoint clusters using k-means. By doing so, we reduce later calculations to a small number of representative viewpoints. The clustered viewpoints are weighted according to the relative durations of the view voxels. We denote the clustered viewpoints $\vec{u}_i \in \mathrm{I\!R}^3, \mathrm{i} \in [1, \mathrm{N_v}]$, and the associated weights $\omega_i \in [0,1]$. The weights ω_i are normalized to the unit interval and represent the relative importance of each viewpoint cluster.

Visibility versus viewpoint data

The visibility measurements along the surfaces are providing information, which can not be inferred with the viewpoints alone. Visibility measures temporary occlusions and not only occlusions of static surfaces. That is, even if many viewpoints have a clear view to a static surface, the visibility of that surface might still be low, for instance, because of occlusions by the user's hand or clutter due to moving physical objects. Lastly, viewpoints do not contain information about the view direction. Instead, the combination of viewpoints and surface visibility ensures that objects are not placed behind users.

OPTIMIZATION

Based on the processed data, we optimize the layout of the developer's virtual scene. The design vector (see *Design vector v_d*) is used as an interface by the optimizer to control the scene state, i. e., postures of virtual objects.

Objective Function

The objective function f_o evaluates the quality of the current scene state. It is given by:

$$f_o : \mathrm{I\!R}^{\mathrm{N_d}} \rightarrow [0,1]$$
$$f_o = Q_P \cdot Q_L$$

The input for the objective function consists of the current design vector values. The output of the objective function is a real number between 0 (lowest quality) and 1 (highest quality). Q_P is the **projection quality** and describes the perceived quality of the projection, given the room geometry, projector poses and a set of user viewpoints. In contrast, the **layout quality** Q_L describes the scene layout quality, independently of the projections on the surfaces. It is defined during the design phase, as described in *Layout quality Q_L*.

Projection quality Q_P

We calculate the projection quality on a per object basis with each virtual object's projection quality $Q_o \in [0,1]$. Given a set of user viewpoints as defined in *Clustered viewpoints $\vec{u}_i$*, Q_o describes the quality of the object's projection from these viewpoints. It is given by

$$Q_o = {q_{ill}}^{\phi_{ill}} \cdot {q_{vis}}^{\phi_{vis}} \cdot {q_{dist}}^{\phi_{dist}}$$

The terms q_{ill}, q_{vis} and q_{dist} are independently calculated quality terms. The exponents ϕ_{ill}, ϕ_{vis} and ϕ_{dist} can be used to adjust the influence of each term.

The illumination term $q_{ill} \in [0,1]$ describes how much of the content can be projected by at least one projector. 0 means that no projector can display any of the projections, since it is outside their field-of-view or in permanent shadow. 1 means that all projections can be displayed at all times.

The visibility term $q_{vis} \in [0,1]$ describes how likely it is that the projections will be visible for users according to their measured view behavior.

The distance term $q_{dist} \in [0,1]$ describes how close the points along the surface of the virtual object are to the physical surface when seen from the user perspective. This accounts for *depth discrepancies* (see Figure 2). To describe q_{dist} more formally, we first define the *distance discrepancy* p_{dist} for one viewpoint $\vec{u}_i$ and one point $\vec{s}_v$ on the virtual surface:

$$p_{dist} = \begin{cases} \frac{||\vec{d}_v||}{||\vec{d}_r||}, & \text{if } ||\vec{d}_v|| \geq ||\vec{d}_r|| \\ \frac{||\vec{d}_r||}{||\vec{d}_v||}, & \text{if } ||\vec{d}_v|| < ||\vec{d}_r|| \end{cases}$$

$$\vec{d}_v = \vec{s}_v - \vec{u}_i, \;\; \vec{d}_r = \vec{s}_r - \vec{u}_i, \;\; \frac{\vec{d}_v}{||\vec{d}_v||} = \frac{\vec{d}_r}{||\vec{d}_r||}$$

Here, the physical surface point $\vec{s}_r$ is the closest intersection point with the physical surface of the line of sight looking from $\vec{u}_i$ through $\vec{s}_v$. As seen in the equation, p_{dist} depends on the ratio of the distances and not on the absolute difference. This accounts for the decrease in depth discrepancies with increasing distance between user and real surface. The resulting distance term q_{dist} integrates p_{dist} over all viewpoints and over all viewpoint-visible points $\vec{s}_v$ along the surface of the virtual object.

The overall projection quality Q_P of the entire scene is the product of all object projection qualities.

$$Q_P = \prod_{Objects} Q_o$$

This implies that a poor score for at least one object leads to a poor score for the entire scene. We provide details on how we calculate the projection quality in the *Projection quality estimation* subsection.

Optimization algorithm

Since we do not have an analytical representation of the objective function, we opt for a gradient-free optimization technique, namely CMA-ES for *covariance matrix adaptation evolution strategy* [11]. The central idea of this algorithm is to sample the parameter space, represented by the space of design vectors $v_d \in \mathbb{R}^{N_d}$ and to fit a multivariate normal distribution to the data. New samples are drawn from this distribution to update the mean and covariance matrix for the next iteration. For details, we refer the reader to [14, 13, 11]. The algorithm's ability to naturally incorporate box constraints (bounds on the elements in v_d) is very convenient in our scenario. CMA-ES has been shown to be superior to competing black-box optimization algorithms for a large class of complex optimization problems [12].

Naturally, CMA-ES works best for convex objective functions that resemble a normal distribution. Locally, that is, when optimizing parameters for projection on a single surface, we observe that this assumption is fullfilled very well. However, in the case of multiple objects and many different potential projection targets, the algorithm might get "stuck" at a local minimum. Moreover CMA-ES is known to underperform for separable functions. If several objects have no interaction terms, then the algorithm cannot leverage the fact that these objects can be optimized independently. For these reasons, a good start configuration is necessary in order to yield good results. We therefore propose a two-phase optimization strategy where good starting points are computed before starting the full optimization.

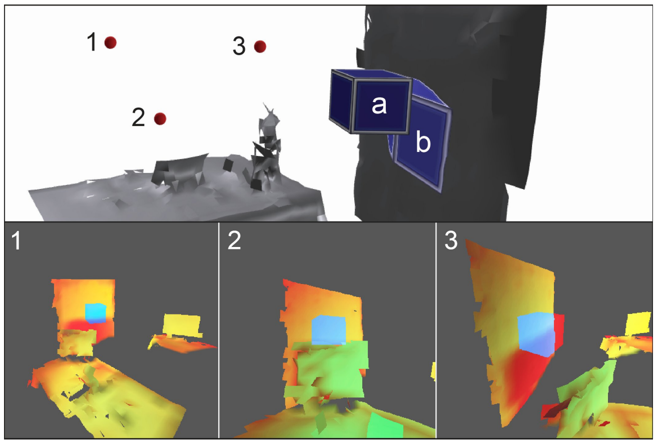


Figure 7. Estimation of the projection quality of one virtual object's current pose. The room's walls are not depicted for the sake of simplicity. (a) is a virtual cube and (b) is one of the projections onto the surface mesh. The surface mesh and the virtual object are rendered from the sampled viewpoints using a shader that calculates the projection quality at each pixel. The pixel colors and transparencies in the three renderings encode the distance from the viewpoint, as well as the illumination and visibility for fast GPU processing. Gray regions are not illuminated. Rendering number (3) of the virtual cube contains pixels that are not projected onto a physical surface (red regions in the cube), which leads to a low overall projection quality.

In the first phase, we sample the objective function for each object individually on a regular grid (using a resolution of 10^3 grid points). At each grid point, we only consider position and scale parameters and evaluate the objective function for 8 scale values that are uniformly distributed on the allowed interval for the scale parameter. The minimum energy value for all optimizations is stored for each grid point. Since we expect optimal object positions to be close to projection targets, we first compute the distance between the grid point and the closest point on the geometry and skip the objective evaluation, if this distance exceeds a certain threshold.

For this grid, we compute local maxima, i. e. points that have an energy value exceeding that of all of its up to 6 neighbors. All these maxima are clustered according to proximity, so as to consolidate multiple close extrema. We call representatives of these clusters *candidates*. There are usually one or more candidates per projection surface and object. Note that we do not consider interaction terms between the objects up to this point.

For the second phase, the candidates for each object serve as starting points for several runs of CMA-ES. For a fixed number of iterations, starting positions for each object are chosen at random from their candidates. The CMA-ES now optimizes for all parameters in the design vector, this time also considering interaction terms like minimal pairwise distance and rotations. The final result is the design vector of highest energy encountered.

Using this two-phase approach, we overcome the problem of local minima by choosing different promising start configurations. An alternative approach would be to adaptively sample the space of design vectors on a regular grid in $\mathbb{R}^{N_d}$ similar to [8]. In our case, however, the dimension of the design space is much larger and a regular sampling strategy becomes inefficient. Our approach combines regular sampling in the first phase with CMA-ES iterations.

IMPLEMENTATION

This section provides some details about our infrastructure, software and our GPU implementation of the projection quality estimation.

Projection quality estimation

To estimate the terms of the projection quality, we render the virtual objects from the viewpoints $\vec{u}_i$ that we clustered during the post-processing phase of the data acquisition. For each object and viewpoint, the mesh surface and the virtual object are rendered, using a virtual camera, which is looking from the viewpoint to the object. We implemented shaders that use RGBA colors to encode illumination, visibility and distance. Figure 7 shows an example. The reconstructed mesh surface and the virtual object are rendered into two different buffers. Note that the surface mesh encodes surface quality data in the UV coordinates. During rendering, three aspects are measured for each pixel:

- **Illumination**. This can be determined from the UV coordinates of the surface mesh at this pixel and is used for calculating q_{ill}.
- **Visibility**. This can be determined from the UV coordinates of the surface mesh at this pixel and is used for calculating q_{vis}.
- **Distance**. The distance of the unprojected pixel in world coordinates to the viewpoint. We apply the inverse model-view-projection matrix to the normalized device coordinates of the pixel which gives us the pixel-position in world-coordinates. This is later used for calculating q_{dist}.

A compute shader combines the renderings to calculate and average q_{ill}, q_{vis} and q_{dist} on a per-pixel basis.

The steps for evaluating one virtual projected object are listed in Figure 8.

Render modifiers

Static virtual 3D objects are only a very basic use for interactive projection mapping environments. To support dynamic content, we implemented a *render modifier* concept. Developers can control how objects are rendered during projection quality estimation. For instance, virtual 3D characters

```
function ProjectionQuality($Object$)
    $q_{ill} = q_{vis} = q_{dist} = 0$                                  ▷ Reset
    $W_{pix} = 0$                                       ▷ Sum of all pixel weights
    for all Viewpoints $\vec{u}_i, i \in [1..N_v]$ do
        Set camera to look from $\vec{u}_i$ to $Object$
        //Render physical surface with shader into first texture
        Clear texture $T_S$ with rgba(0, 0, 0, 0)
        Render $MeshSurface$ into texture $T_S$ with shader:
        for all Pixels $P^S$ to render in $T_S$ do
            $P^S$:red = $||P^S_{cam}||$                      ▷ Distance to viewpoint
            $P^S$:alpha = UV.x                                   ▷ Illumination
            $P^S$:green = UV.y                                     ▷ Visibility
        end for
        //Render virtual object with shader into second texture
        Clear texture $T_V$ with rgba(0, 0, 0, 0)
        Render $Object$ into texture $T_V$ with shader:
        for all Pixels $P^V$ to render in $T_V$ do
            $P^V$:red = $||P^V_{cam}||$                      ▷ Distance to viewpoint
            $P^V$:alpha = $\omega_i$                             ▷ Viewpoint weight
        end for
        //Combine, calculate and add up with compute shader
        for all $(P^S_j, P^V_j)$, $j \in [1..\#pixels]$ do
            $w_{pix} = P^V_j$:alpha                                  ▷ Pixel weight
            $q_{ill}$+= $P^S_j$:alpha $\cdot w_{pix}$
            $q_{vis}$+= $P^S_j$:green $\cdot w_{pix}$
            $q_{dist}$+= $(P^S_j\text{:red} > P^V_j\text{:red} ~?~ \frac{P^V\text{:red}}{P^S\text{:red}} : \frac{P^S\text{:red}}{P^V\text{:red}}) \cdot w_{pix}$
            $W_{pix}$ += $w_{pix}$
        end for
    end for
    $q_{ill}$ /= $W_{pix}$
    $q_{vis}$ /= $W_{pix}$
    $q_{dist}$ /= $W_{pix}$
end function
```

Figure 8. Simplified steps for estimating the illumination, visibility and distance for one projected object using multiple viewpoints. Note that the pixel weight w_{pix} equals zero where no virtual pixel is drawn.

or creatures are typically animated. To account for this, a render modifier renders multiple animation states of the animation into the same viewpoint rendering (see Figure 9.a). Furthermore, render modifiers can adapt to the current sample viewpoint during projection quality estimation. In interactive projection mapping environments, objects adapt to the user's viewpoint not only in terms of perspective correction, but possibly also for more specialized behavior. For instance, content can be programmed, so as to always face the viewer, which is common for perspective corrected text (see Figure 9.b) or spatialized desktop windows. More complex use cases are virtual articulated objects, which change their posture, depending on the viewing angle (see Figure 9.c). Besides these default render modifiers, developers can define any render modifier to improve the accuracy for the projection quality estimation and hence for the optimization.

Hardware and software

Our hardware consists of multiple *Kinect* devices, each connected to an *Intel NUC Mini-PC*. Depth and infrared data is streamed via Ethernet to the central PC. We combine the Kinect streams similar to [23]. Furthermore, we downsample the streams and use the depth compression algorithm by Wilson [27] to reduce network traffic. Our framework also handles projection mapping and other rendering-related functionalities. The *OptiSpace Data Acquisition Application* for processing the 3D data is implemented in *Unity*.

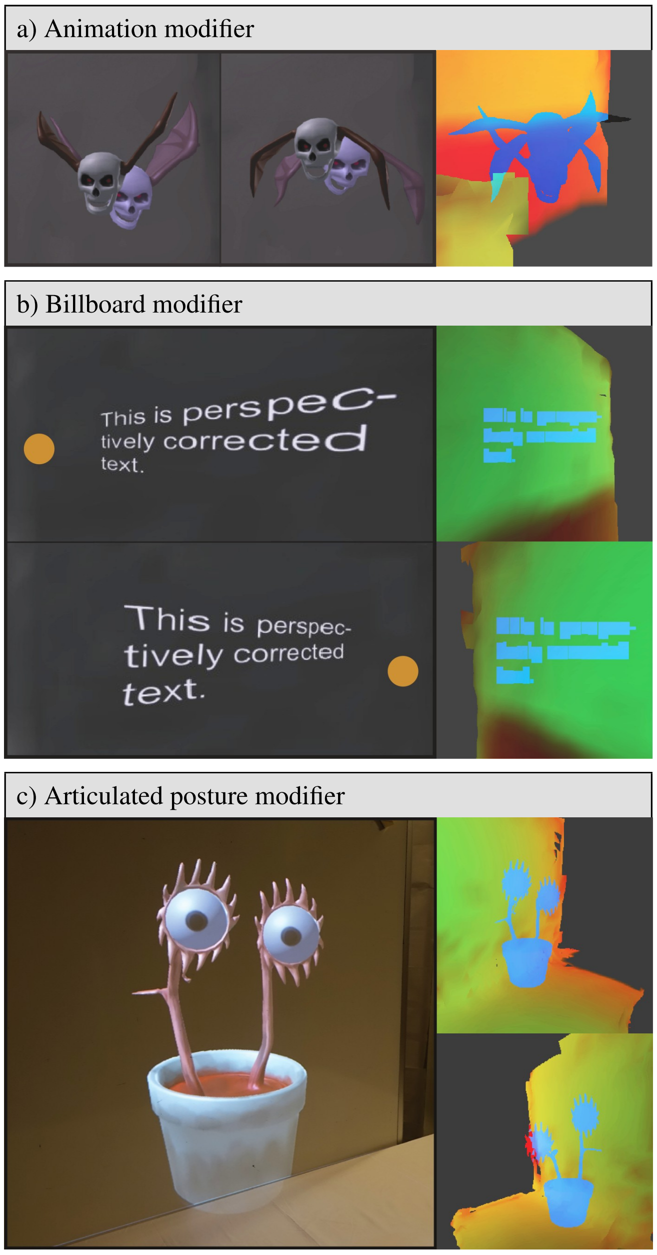

a) Animation modifier
b) Billboard modifier
This is perspec-tively corrected text.
This is perspec-tively corrected text.
c) Articulated posture modifier

Figure 9. Example render modifiers. (a) Projection quality for an animated object (a wing-flap motion) on a whiteboard. Rendering multiple animation states at once (right) reveals that parts of the object are not projected at certain points in time during animation. (b) Perspectively corrected text always faces the viewer (orange sphere). (c) The creature always looks at the viewer and adjusts its articulated posture accordingly. This is taken into account when projecting from different viewpoint samples (right).

The framework for developers is implemented as a *Unity* plugin. Developers create *Unity GameObjects* and attach components according to their desired properties for optimization. An additional GUI and visualization tool built as a *Unity Editor* plugin allows for adjustments and the testing of design vectors and layout quality functions. Furthermore, we provide abstract C# classes to extend the set of components for optimization. For instance, if developers want to add further design vector elements, they need to create a mapping between scene properties and the design vector entries. One example is a design vector entry, which controls a non-rigid shape of an object.

Our optimization algorithm is implemented as a native C++ plugin. The algorithm builds on top of the *cmaes* library [3]. The projection quality is implemented as a set of *NVIDIA CG* shaders and a compute shader for fast parallel processing of viewpoints.

Performance

To increase performance, we dynamically adapt the accuracy of the projection quality estimation. That is, for the initial search of candidate positions, we reduce the resolution of the renderings and number of viewpoints (currently 4 viewpoints with a 128x128 resolution). For detailed placement around the candidate placements, we increase the resolution as high as 512x512 and 16 viewpoints.

The time needed for the optimizer to find an optimal scene layout depends on various factors. Depending on the complexity of the scene to be optimized, we render between 100000 and 500000 viewpoints. Finding an optimal placement usually takes around 1 to 2 minutes.

RESULTS

In order to explore how well the system adapts content to different target environments, we conducted a small trial. We invited 10 participants to do a one hour working session with different room layouts. Participants were asked to bring their laptop and work as they normally would. There was no supervision and participants simply performed their everyday work tasks. We created two virtual scenes. The first is a perspectively corrected notification, and the second consists of three virtual creatures. We recorded participants and acquired data as described in the *Data acquisition* section. A few examples are shown in Figure 10.

- **Figure 10.1**: This participant generated many viewpoint samples. The notification is placed at the canvas in front of him.
- **Figure 10.2**: Even though clearly visible, the temporary shadows (dark shades) render most parts of the whiteboard a poor display surface. The notification is placed at a location where the participant did not cause projector shadows.
- **Figure 10.3**: This participant worked at the table. The optimizer positioned the virtual plant on the physical table at the wall.

The results make us confident that OptiSpace can find suitable virtual scene layouts for diverse target environments.

EXTENSIONS

In order to explore the broader potential of our approach, we have developed a number of extensions to our system. To further utilize the measured data, we implemented a runtime API, i. e., OptiSpace can run as part of the target application after the initial optimization. The runtime API has two parts: *online measurements* and *data access*.

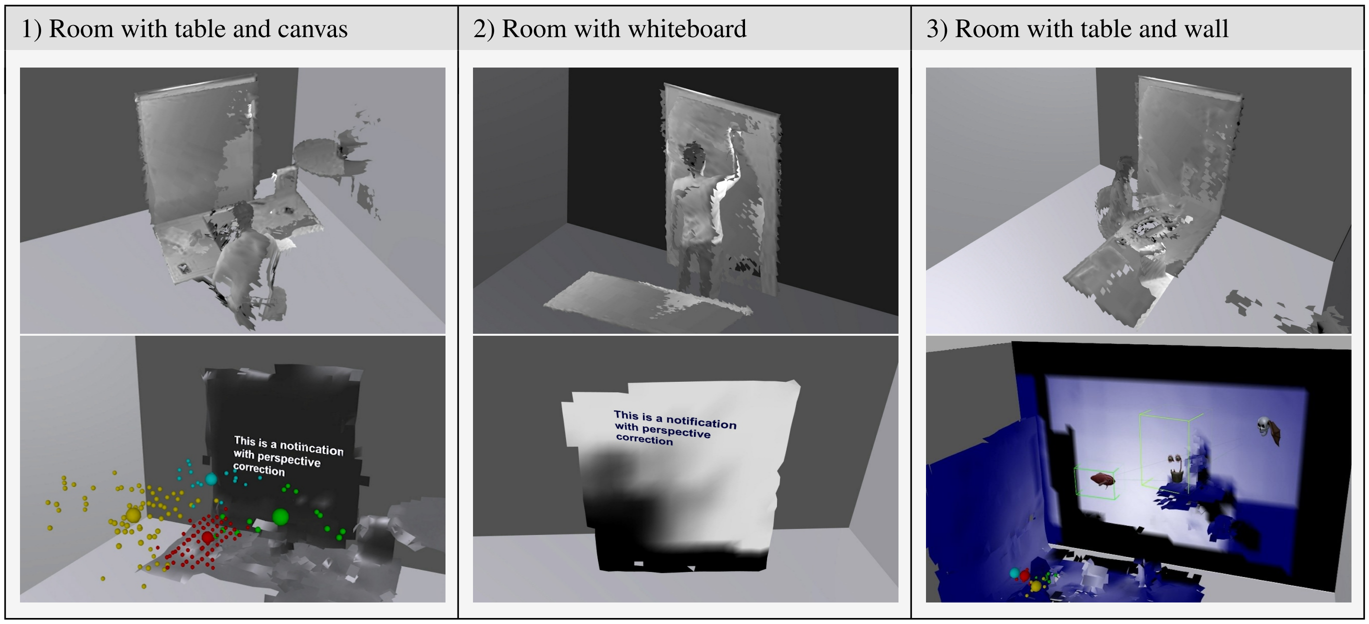

1) Room with table and canvas
2) Room with whiteboard
3) Room with table and wall
This is a notincation with perspective correction
This is a notification with perspective correction

Figure 10. We invited several participants to test our system with different room layouts and viewing behaviors. The first row shows the different target environments. The second row shows how OptiSpace adapts the virtual scene to this target environment.

Online measurements

Instead of or in addition to measuring viewpoints, illumination and visibility in the data acquisition phase, the values can be updated during runtime. A specialized runtime measurement routine is designed to meet real-time requirements. Instead of maintaining a voxel grid, we directly update the values on the reconstructed static surface mesh. For instance, we measure projector shadows on the mesh and update the values for brightness in real-time.

Real-time quality sensing

Besides optimization, we allow for low-level access to the measurement data during runtime - regardless of whether the data is still updated online. Furthermore, we provide higher-level functionalities called *quality sensing objects* (see Figure 11). An object can access the terms of the projection quality and the gradients locally to react accordingly in real-time. Figure 12 shows some examples for quality-sensing objects.

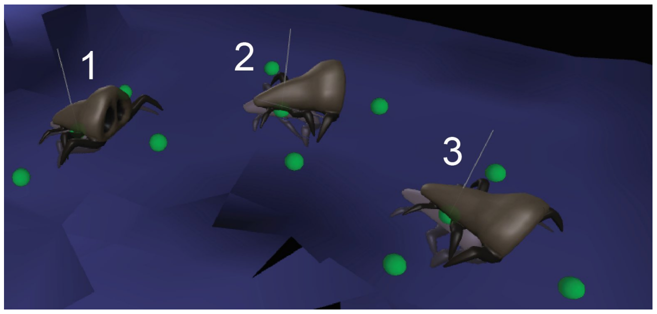


Figure 11. Example of a quality-sensing interactive object. The sensors depicted in green evaluate the projection quality around the object so as to adjust the movement direction. The creature crawls from (1) to (3) around a dark spot, where it would not be projectable, due to a dark object being on the table.

DISCUSSION AND FUTURE WORK

OptiSpace enables content developers for projection mapped augmented reality to develop content independently of the room geometry where the content is to be shown. Previously, content usually had to be carefully adjusted to a specific room geometry, and was not directly reusable for other rooms. With OptiSpace, the same content can be reused without changes. This improvement in the scalability of projection mapping content might eradicate a significant factor constraining projection mapped augmented reality.

While we have tested OptiSpace with 10 different room layouts and users, evaluating it with multiple content developers remains to be done. In particular, it would be interesting to see how easily content developers can adapt to describing their intentions in the design vector and layout-quality function.

More generally, we see OptiSpace as an example of interfaces that simulate the perceptions of the user and adapt accordingly. Such interfaces could create a model of the visual scene that users perceive, including user perspective and real and virtual objects. They would then render the scene users would perceive if the system were to behave in a certain way. They would evaluate this perception and optimize the system actions accordingly. We believe such interfaces could contribute greatly to human-computer interaction by making interfaces adapt not only to actual user behavior, but also to their physical surroundings. Such interfaces could adapt not only content location, but also motion, shapes, colors, brightness, contrast, etc. Thus, they could blend much better into their visual environment, not only in the case of augmented reality, but possibly also for more conventional desktop and mobile interfaces. One particular example of how this approach could be used would

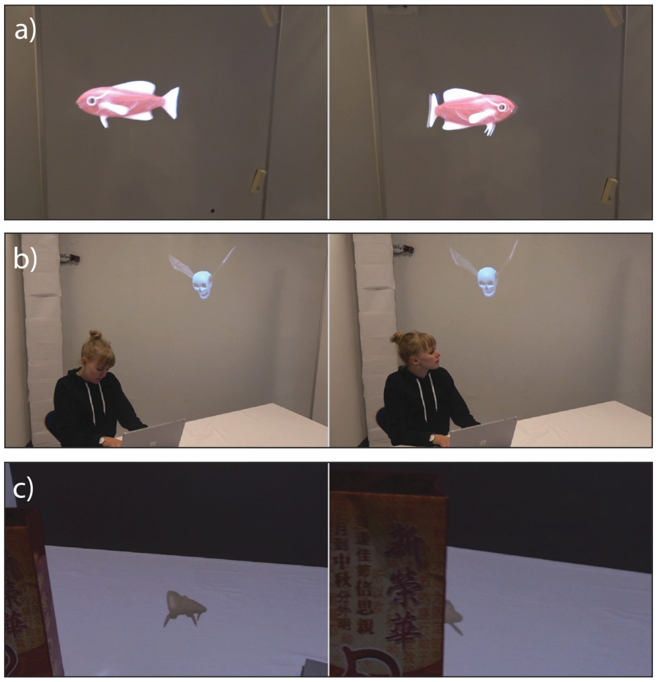


Figure 12. Examples of quality-sensing objects utilizing different terms of Q_P. (a) The fish swims within the whiteboard where it can be projected. The visibility is ignored. (b) The skull always moves so that it is outside the field of view of the user in the picture, but can be projected for the user behind the camera. (c) Whenever it becomes visible, the creature crawls into hidden places. It searches for locations with high projection quality, but low visibility.

be implementing the vision of change-blind information displays [16]. Updated and changed displayed information in ubiquitous computing environments might capture attention and thus disrupt users. Using a system like OptiSpace, we could keep information static whenever users see it and only update it when users cannot see it, e.g., when it is temporarily occluded. This approach would exploit the effect of change blindness and reduce the number of distractions in ubiquitous computing environments.

CONCLUSION

We have presented OptiSpace, a system for optimizing the placement of interactive projection mapping content, based on empirical user behavior. Developers implement interactive projection mapping applications just once, without knowledge of the actual room geometry or possible user viewing angles. Applications can then be deployed in different uncontrolled environments, not necessarily by the developers themselves. OptiSpace automatically measures the target environment, including users, who are completely uninstrumented. Our optimization is based on our measurements and various programmable attributes and behaviors. We have proposed an approach to estimating the quality of perspectively corrected content. The generic design of our architecture makes it applicable to a broad range of dynamic interactive projection mapping applications for uncontrolled environments.

ACKNOWLEDGEMENTS

We want to thank David Lindlbauer for his valuable feedback. This work has been supported by IFD grant no. 3067-00001B for the project entitled: MADE - A platform for future production.

REFERENCES

1. 2014. The other resident. `https://www.youtube.com/watch?v=NXxVXQYlSXc`. (2014). Accessed: 2018-01-08.
2. Blaine Bell, Steven Feiner, and Tobias Höllerer. 2001. View Management for Virtual and Augmented Reality. In *Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technology (UIST '01)*. ACM, New York, NY, USA, 101–110. DOI: `http://dx.doi.org/10.1145/502348.502363`
3. Emmanuel Benazera. 2015. libcmaes. (2015). `https://github.com/beniz/libcmaes` Accessed: 2017-09-19.
4. Oliver Bimber, Andreas Emmerling, and Thomas Klemmer. 2005. Embedded entertainment with smart projectors. *Computer* 38, 1 (2005), 48–55.
5. Oliver Bimber, Daisuke Iwai, Gordon Wetzstein, and Anselm Grundhöfer. 2008. The Visual Computing of Projector-Camera Systems. In *Computer Graphics Forum*, Vol. 27. Wiley Online Library, 2219–2245.
6. Oliver Bimber and Ramesh Raskar. 2005. *Spatial augmented reality: merging real and virtual worlds*. CRC press.
7. Daniel Cotting and Markus Gross. 2006. Interactive environment-aware display bubbles. In *Proceedings of the 19th annual ACM symposium on User interface software and technology*. ACM, 245–254.
8. Andreas Fender, David Lindlbauer, Philipp Herholz, Marc Alexa, and Jörg Müller. 2017. HeatSpace: Automatic Placement of Displays by Empirical Analysis of User Behavior. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*. ACM, 611–621.
9. Markus Funk, Thomas Kosch, Katrin Wolf, Pascal Knierim, Sven Mayer, and Albrecht Schmidt. 2016. Automatic Projection Positioning Based on Surface Suitability. In *Proceedings of the 5th ACM International Symposium on Pervasive Displays (PerDis '16)*. ACM, 75–79.
10. Anselm Grundhofer and Oliver Bimber. 2008. Real-time adaptive radiometric compensation. *IEEE transactions on visualization and computer graphics* 14, 1 (2008), 97–108.
11. Nikolaus Hansen. 2006. An analysis of mutative σ-self-adaptation on linear fitness functions. *Evolutionary Computation* 14, 3 (2006), 255–275.
12. Nikolaus Hansen, Anne Auger, Raymond Ros, Steffen Finck, and Petr Pošík. 2010. Comparing Results of 31

Algorithms from the Black-box Optimization Benchmarking BBOB-2009. In *Proceedings of the 12th Annual Conference Companion on Genetic and Evolutionary Computation (GECCO '10)*. ACM, 1689–1696.

13. Nikolaus Hansen and Stefan Kern. 2004. Evaluating the CMA Evolution Strategy on Multimodal Test Functions. In *Parallel Problem Solving from Nature PPSN VIII (LNCS)*, X. Yao et al. (Eds.), Vol. 3242. Springer, 282–291.

14. Nikolaus Hansen and Andreas Ostermeier. 2001. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation* 9, 2 (2001), 159–195.

15. John Hardy, Carl Ellis, Jason Alexander, and Nigel Davies. 2013. Ubi displays: A toolkit for the rapid creation of interactive projected displays. In *The International Symposium on Pervasive Displays*.

16. Stephen S Intille. 2002. Change blind information display for ubiquitous computing environments. In *UbiComp 2002: Ubiquitous Computing*. Springer, 91–106.

17. Brett Jones, Rajinder Sodhi, Michael Murdock, Ravish Mehra, Hrvoje Benko, Andrew Wilson, Eyal Ofek, Blair MacIntyre, Nikunj Raghuvanshi, and Lior Shapira. 2014. RoomAlive: Magical Experiences Enabled by Scalable, Adaptive Projector-camera Units. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology (UIST '14)*. ACM, 637–644.

18. Hanhoon Park, Moon-Hyun Lee, Byung-Kuk Seo, Hong-Chang Shin, and Jong-Il Park. 2006. Radiometrically-compensated projection onto non-lambertian surface using multiple overlapping projectors. *Advances in Image and Video Technology* (2006), 534–544.

19. Tomislav Pejsa, Julian Kantor, Hrvoje Benko, Eyal Ofek, and Andrew Wilson. 2016. Room2Room: Enabling life-size telepresence in a projected augmented reality environment. In *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing*. ACM, 1716–1725.

20. Ramesh Raskar, Greg Welch, Matt Cutts, Adam Lake, Lev Stesin, and Henry Fuchs. 1998. The office of the future: A unified approach to image-based modeling and spatially immersive displays. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*. ACM, 179–188.

21. Jan Riemann, Mohammadreza Khalilbeigi, Martin Schmitz, Sebastian Döweling, Florian Müller, and Max Mühlhäuser. 2016. FreeTop: Finding Free Spots for Projective Augmentation. In *Proceedings of the 34rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems (CHI EA'16)*. ACM, New York, NY, USA.

22. Makoto Sato and Kaori Fujinami. 2014. Nonoverlapped view management for augmented reality by tabletop projection. *Journal of Visual Languages & Computing* 25, 6 (2014), 891–902.

23. Maurício Sousa, Daniel Mendes, Rafael Kuffner Dos Anjos, Daniel Medeiros, Alfredo Ferreira, Alberto Raposo, João Madeiras Pereira, and Joaquim Jorge. 2017. Creepy Tracker Toolkit for Context-aware Interfaces. In *Proceedings of the 2017 ACM International Conference on Interactive Surfaces and Spaces (ISS '17)*. ACM, 191–200.

24. Jeroen Van Baar, Thomas Willwacher, Srinivas Rao, and Ramesh Raskar. 2003. Seamless multi-projector display on curved screens. In *Proceedings of the workshop on Virtual environments 2003*. ACM, 281–286.

25. John Vilk, David Molnar, Eyal Ofek, Chris Rossbach, Ben Livshits, Alex Moshchuk, Helen Wang, and Ran Gal. 2014. *SurroundWeb: Least Privilege for Immersive "Web Rooms"*. Technical Report.

26. Gordon Wetzstein and Oliver Bimber. 2007. *Radiometric Compensation through Inverse Light Transport*. Technical Report. Juniorprofessur Augmented Reality. `http://nbn-resolving.de/urn:nbn:de:gbv:wim2-20111215-8126`

27. Andrew D Wilson. 2017. Fast Lossless Depth Image Compression. In *Proceedings of the 2017 ACM International Conference on Interactive Surfaces and Spaces*. ACM, 100–105.

28. Andrew D Wilson and Hrvoje Benko. 2010. Combining multiple depth cameras and projectors for interactions on, above and between surfaces. In *Proceedings of the 23nd annual ACM symposium on User interface software and technology*. ACM, 273–282.