Screen Space 3D Diff: A Fast and Reliable Method for Real-time 3D Differencing on the Web

Jozef Doboš
3D Repo Ltd
London, UK
ceo@3drepo.org

Carmen Fan
3D Repo Ltd
London, UK

Sebastian Friston
University College London
London, UK

Charence Wong
3D Repo Ltd
London, UK

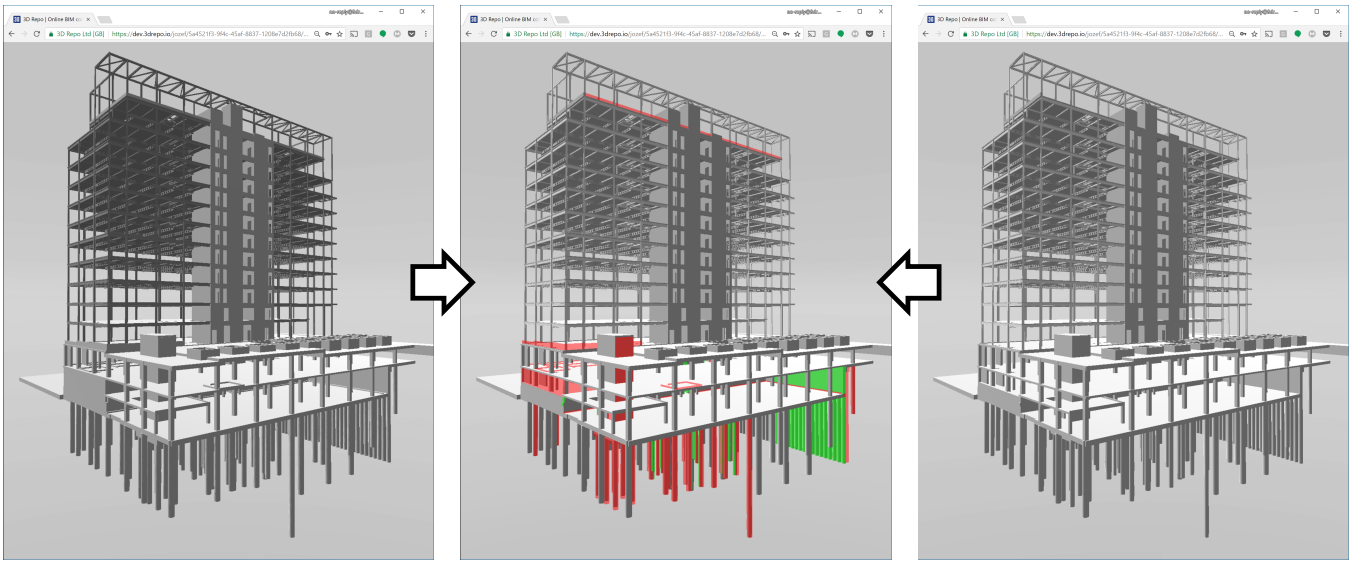


Figure 1: Revisions of a construction 3D model (left/right) are differenced in screen space on a web browser in real time. Resulting visualization (middle) shows deletions in red and additions in green. Model courtesy of Canary Wharf Contractors.

ABSTRACT

We introduce Screen Space 3D Diff, an interactive method for fast and reliable visual differencing on the web. This method is based on the properties of 3D scenes in 2D space such as depth, color, normals, UV, texture, etc. The central idea states that if two objects project into the same screen space with identical properties, they can be assumed to be identical; otherwise, they are different. In comparison to previous works that require large computational overheads in 3D space, our approach is significantly easier to implement and faster to determine disparities. This is because screen space methods scale with resolution and not scene complexity. They also lend themselves to massive parallelization on modern GPU hardware, and have the added advantage of being accurate at pixel level regardless of magnification. The combination of these benefits allows for instant real-time differencing with minimal disruption to the rendering pipeline even on web browsers. We demonstrate two different implementations of this method; one in a desktop application and one in Unity 3D game engine on the web. The performance of the proposed method across a range of devices is presented and practical benefits of our approach are further evaluated in a user study of twenty professionals with several large-scale models that are typical of the Building Information Modelling paradigm. Based on this evaluation we conclude that our method is invaluable in day-to-day engineering workflows and is well suited to its purpose.

CCS CONCEPTS

• **Computing methodologies** → *Rasterization*; *Image processing*;

KEYWORDS

differencing, screen space, z-buffer, Unity 3D

ACM Reference Format:
Jozef Doboš, Carmen Fan, Sebastian Friston, and Charence Wong. 2018. Screen Space 3D Diff: A Fast and Reliable Method for Real-time 3D Differencing on the Web. In *Web3D '18: Web3D '18: The 23rd International Conference on Web3D Technology, June 20–22, 2018, Poznan, Poland*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3208806.3208809

1 INTRODUCTION

Being able to quickly and reliably establish what has changed between two revisions of the same 3D scene, or even between two entirely different 3D scenes, on the web has numerous applications across various industries. For example, Computer Aided Design (CAD) and Building Information Modelling (BIM) rely heavily on 3D models that change over time as the designs evolve. The

Web3D '18, June 20–22, 2018, Poznan, Poland
2018. ACM ISBN 978-1-4503-5800-2/18/06…$15.00
https://doi.org/10.1145/3208806.3208809

myriad of editing tools and ever growing collaborative teams make keeping track of frequent changes in 3D space extremely difficult.

Despite being the norm in software engineering, cloud-based version control and change detection are still not prevalent in 3D modelling. Only a few tools such as Autodesk Maya or Vistrails record all modelling history by instrumenting the editor. If, however, no record of the editing provenance exists, it is nearly impossible to reliably detect all discrepancies between the models, let alone do so quickly using the existing techniques. It is well known in the industry that exchanging models between different software packages can result in information loss due to incompatibilities. In addition, many packages can exhibit "vendor lock-in", whereby users are unable to easily exchange information contained within proprietary file formats. Whilst open standards such as Collaborative Design Activity (COLLADA) [2012], Industry Foundation Classes (IFC) [2013] (originally based on the Standard for the Exchange of Product model data (STEP) [1994]), and AutoCAD's Drawing Exchange Format (DXF) do exist, the full support for their respective encodings is still largely inconsistent. The proposed solution is a robust, file-format-independent 3D differencing method that can be used in multiple situations, even on web browsers. For real-world industrial applications, a successful solution should be i) consistent and accurate; ii) work across different tools regardless of data representation; iii) fit seamlessly into existing editing pipelines; and iv) scale to large datasets without significant overhead. The tractability is especially important as large-scale industrial 3D scenes can often span hundreds of thousands of separate components.

Contributions. To solve the aforementioned problems, we present a novel *Screen Space 3D Diff* (see Fig. 1) method for visual 3D differencing. Unlike existing methods listed in §2.2, our solution is based on screen space differencing which scales with resolution and not scene complexity. It also has the added benefit that it provides pixel-level accuracy at any magnification level. Thus, our contributions can be summarized as follows:

(1) A novel, fast and reliable method for real-time visual 3D Diff;
(2) Addition of differences in colors, textures, normals, etc.;
(3) Desktop as well as web-based implementation;
(4) A comparison to previous work on a selection of 3D models;
(5) A user study with industrial professionals that demonstrates its usefulness in everyday engineering workflow.

2 RELATED WORK

The process of *differencing*, i.e. finding the commonalities and discrepancies between two or more inputs, has long been the focus of academic as well as industrial research. Traditionally, the topic was divided between 2D and 3D differencing. In this paper we attempt to blur the boundaries between such approaches by relying on 2.5D screen space representation of a 3D scene.

2.1 2D Differencing

Chen et al. [2011] introduced non-linear version control for 2D images. There, a directed acyclic graph (DAG) represents images and edits evolving over time. Images are compared by superimposing one over the other with support for operation-based merging. A similar approach to image comparison is offered by a number of tools including Guiffy Image Diff Tool, Araxis Merge and GitHub Image View, which overlay images and display commonalities and differences with support for overlays, side-by-side visualizations, sliders, etc. Our method takes inspiration from these and applies a similar approach to a 3D view frame by frame.

2.2 3D Differencing

Matching identifiers. Doboš and Steed [2012a] devised an interactive *3D Diff* tool which assumes component-level correspondence based on matching identifiers (IDs) stored in a version control system 3D Repo [2012b]. This includes visualization modes such as *superposition*, *juxtaposition*, and *explicit encodings*, as well as *two*, *three* and *n-way* differencing. Similarly, long established desktop-based solutions such as *Detect IFC Model Changes* in ArchiCAD by Graphisoft, *IFC Change Management* in Tekla by Trimble and Model Compare in FME by Safe rely on object-level IDs being preserved between consecutive revisions of 3D files. Online, *Version Compare* in BIM 360 Team by Autodesk is based on Three.js library and supports change detection in 2D as well as 3D using matching IDs. Equivalent solutions also exist in cost estimation packages such as Vico Office by Trimble in 3D and CostX by Exactal in 2D. What all these tools have in common is that they are unable to detect changes when the component identifiers are lost or modified. They also take a considerable amount of time to calculate their results. In contrast, our method requires neither matching IDs nor any prior version control knowledge and runs in real-time online.

Voxelization. Other methods rely on implicit correspondence by subdividing the space into a voxel grid. Voxels that share the same (x, y, z) coordinates are implicitly correspondent assuming the input models occupy the same volume. For instance, *Ngrain 3D Geometric Differencing* performed voxelization of the entire rendering volume, which is slow and memory consuming. Similar methods on point-clouds are also employed in industrial 3D laser scanning by Leica and Faro. In contrast, our method is akin to implicit on-the-fly voxelization of the surface layer from the current camera view only. This significantly reduces computational complexity and memory footprint, making it suitable for real-time rendering. It also increases the perceived precision of the differencing as the implicit comparison volume is always equal to the current pixel size.

Editor instrumentation. Another approach to change detection is to record and store the entire editing provenance during the modeling process. Denning et al. [2011] obtained editing sequences using a Blender plug-in. These were analyzed and clustered to convey the changes visually. Similarly, commercial tools, such as Autodesk Maya and VisTrails, record the editing history. In contrast, our method is able to detect differences without pre-recorded histories even when the models were created in different tools altogether.

Correspondence estimation. There also exist those 3D differencing methods that try to establish correspondence across models simply from geometry without the reliance on preserved editing histories or metadata, e.g. MeshGit [2013] and 3D Timeline [2014]. MeshGit generates a dual graph of vertices and edges in order to define a minimum mesh edit distance. From these, an iterative greedy algorithm with backtracking attempts to calculate graph isomorphism. This is non-trivial to implement and takes considerable amount of time to calculate, see comparison to our method

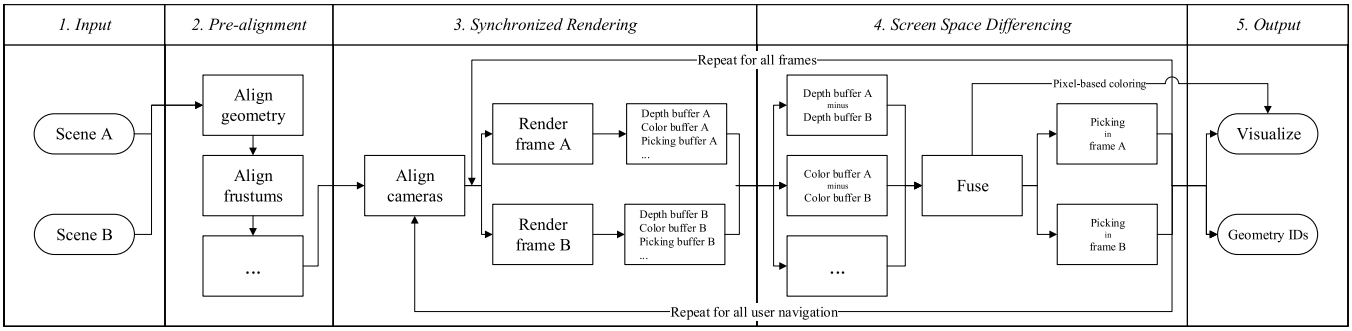


Figure 2: Processing pipeline. Two input 3D scenes A and B are firstly pre-aligned. Next, each scene is rendered from the same camera position in order to produce depth, color, etc. buffers. These are then differenced and finally visualized. The rendering loop is repeated on each frame based on user navigation.

in Fig. 11. GitHub also provides a basic *3D File Diff* visualization for StereoLithography (STL) file comparison. This is achieved by performing Constructive Solid Geometry (CSG) operations on the STL files using binary space partitioning trees and calculating their per-component differences. Removed and added components are cached and displayed in red and green respectively. In addition, a slider enables the users to blend between two revisions seamlessly akin to key-frame animation. A similar slider interface was also employed by Jain et al. [2012] to swap parts of models in order to generate novel shapes as well as in Doboš et al. [2014] to interpolate component changes between consecutive model revisions. There also exist mesh descriptors such as [Han et al. 2017]. Nevertheless, these methods provide only approximate correspondence and do not scale well to real-life industrial applications.

2.3 Screen Space Methods

The biggest advantage of screen space methods, such as Screen Space Ambient Occlusion [Mittring 2007; Shanmugam and Arikan 2007], is that they work in real-time with minimal overhead even on large scenes. Such methods often employ the z-buffer which represents rasterized depth values. Using this, [2002] calculated an error metric to define the dissimilarity between two 3D models. Depth readings from multiple camera views that are perpendicular to the surface of a model (surface roving) are averaged and compared. Similarly, [Krivokuća et al. 2012] defined an error metric based on the depth buffer by rendering 3D models from various viewpoints and averaging the depth discrepancy in order to quantify shape distortion. The closest to our method is Kahn [2013] where 3D models are iteratively aligned with images from a depth camera to provide better visualization for augmented reality. Improvements in alignment are measured by comparing the depth of a synthetic image with the depth from a physical camera at the same pixel.

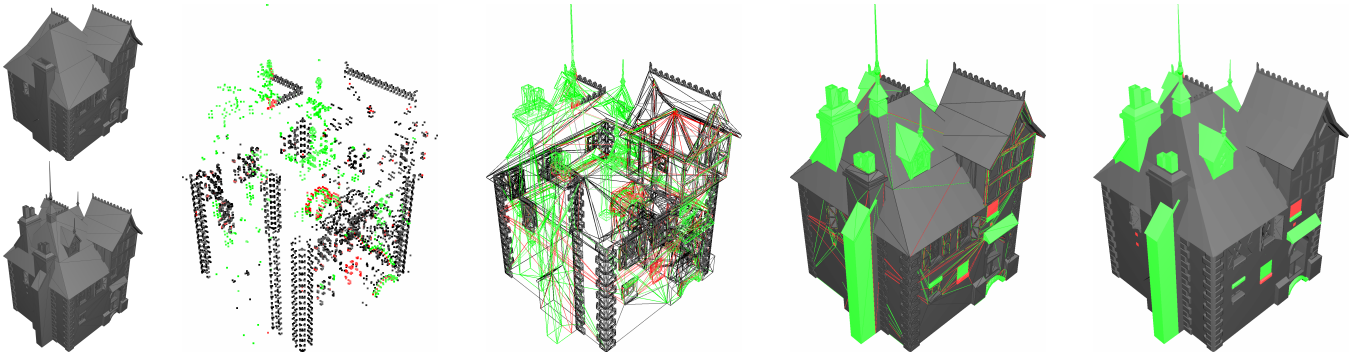
3 METHOD OVERVIEW

Our method computes accurate and consistent visual differences between two 3D scenes as part of a wider 3D Repo web platform [2012b]. Since its initial introduction in 2012, this open source project underwent several major upgrades; firstly with rendering in XML3D [2010] by Doboš et al. [2013], later in X3DOM [2009] by Scully et al. [2015; 2016], and finally in Unity 3D by Friston et al. [2017]. Over the years, 3D Repo has been utilised on some of the largest construction projects in the UK. Especially in the architecture, engineering and construction (AEC) industry, 3D models consist of millions of disparate components and change frequently throughout various stages of a project. Thus, instead of exhaustively calculating differences between components—which would be computationally infeasible—we devise a method that can visualize changes instantly on desktops and web browsers alike.

3.1 Processing Pipeline

To achieve real-time 3D differencing, we use a multi-stage rendering pipeline as shown in Fig. 2. Firstly, the pre-processing step aligns two input scenes in screen space. If the scenes are not in a common coordinate system, they need to be aligned either manually or automatically through a suitable alignment algorithm, e.g. [Aiger et al. 2008]. In addition, the rendering settings such as viewing frustum and projection parameters need to be matched. Next, both scenes are rendered and compared to highlight any geometric and other disparities between them. Finally, as this entire pipeline can be computed in real-time, the rendering settings of both scenes are kept synchronized to allow for reliable comparison on each frame.

Difference metrics. The differences are computed by calculating the distance between separate rendering passes with varying metrics, one at a time. Initially, *depth* differences establish geometrical correspondence across the scenes. This is achieved by comparing pixel-by-pixel signed depth values from their respective z-buffers regardless of the underlying rendering, as shown in Fig. 3. Intuitively, it can be inferred that if the distance to the camera is larger in the first scene than in the second, new parts must have had come into the view, hence, they can be labelled as *added*. Conversely, if the distance to the camera is smaller in the first scene, some parts must have had been omitted, hence *deleted*. Note, however, that the same geometric changes can be also achieved by simply repositioning existing objects instead of actual addition or deletion as discussed in §7. Nevertheless, for any parts of screen space with the same depth, we assume it is the same object and thus further compared on other metrics, namely *color*, *texture*, *normals*, and *metadata*. Each secondary metric can be encoded in a RGBA buffer which is akin to an n-dimensional vector field where $n \leq 4$. To find the distance



(a) (b) (c) (d) (e)

Figure 3: Screen Space 3D Diff. Two input models in (a) are instantly differenced and colored as (b) point cloud, (c) wireframe, (d) shaded wireframe and (e) shaded only rendering. Green corresponds to features being *added* while red *deleted*. Notice how this method correctly identifies differences in geometry, triangulation, vertices and even surface features down to individual pixels beyond simple object-level detection. Any other rendering and shading is equally possible.

between corresponding pixels in the rendered frames A and B, the Euclidean distance between the vectors is calculated as,

$$\Delta\mathcal{M}_{AB} = \left\|\vec{\mathcal{M}}_A - \vec{\mathcal{M}}_B\right\| = \sqrt{\sum_{i=1}^{n}\left(\mathcal{M}_A^i - \mathcal{M}_B^i\right)^2} \tag{1}$$

where $\vec{\mathcal{M}}_A$ and $\vec{\mathcal{M}}_B$ are the corresponding metric vectors in A and B respectively and $\Delta\mathcal{M}_{AB}$ is their difference. Once the distance for every corresponding pixel has been established, a semantic meaning can be assigned to each as,

$$\Delta\mathcal{M}_{AB} := \begin{cases} \textit{added}, & \text{if } \vec{\mathcal{M}}_A = \emptyset \wedge \vec{\mathcal{M}}_B \neq \emptyset \\ \textit{deleted}, & \text{if } \vec{\mathcal{M}}_A \neq \emptyset \wedge \vec{\mathcal{M}}_B = \emptyset \\ \textit{modified}, & \text{if } \vec{\mathcal{M}}_A \neq \emptyset \wedge \vec{\mathcal{M}}_B \neq \emptyset \wedge \left\|\vec{\mathcal{M}}_A - \vec{\mathcal{M}}_B\right\| > \epsilon \\ \textit{unmodified}, & \text{otherwise,} \end{cases} \tag{2}$$

where $\emptyset$ denotes a `null` vector, i.e. value not present, and ϵ a small positive threshold defined for each metric.

4 SCREEN SPACE 3D DIFF

Assuming the input 3D scenes are synchronized, any suitable screen space metric can be used to calculate the differences between them.

4.1 Depth Difference

To reliably compare two depth buffers as shown in Fig. 4, their values must increase linearly with the distance from camera. If it were not the case, the differences would be view dependent, which leads to inconsistent results between subsequent frames as objects get closer or further away from the camera. For a near and far plane, n and f, a normalized depth value z in the range of $[0, 1]$ will project under the standard perspective equation:

$$z' = \frac{f+n}{2(f-n)} + \frac{1}{z}\left(\frac{-fn}{f-n}\right) + \frac{1}{2}. \tag{3}$$

The inversion of this equation transforms from perspectively projected depth value z' to linearized value z'', and takes the form:

$$z'' = \frac{2n}{f+n-(2z'-1)(f-n)}. \tag{4}$$

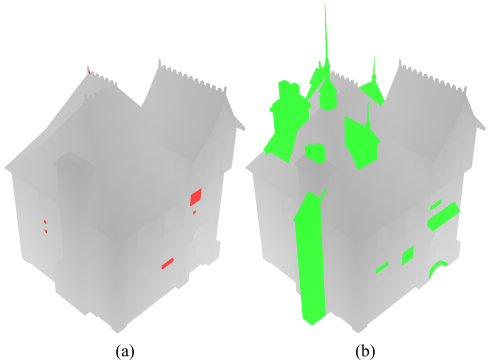
Fig. 5 demonstrates how a consistent semantic meaning can be assigned to all (z''_A, z''_B) corresponding ordered pairs. The case of $z'' = 1$ can represent both an object at the far plane, or simply no object at all. This duality means the semantic assignment from Eq. (2) has to be adapted to utilize the signed depth difference as,

$$\Delta Z_{AB} := \begin{cases} \textit{added}, & \text{if } (z''_A - z''_B) > 0 \\ \textit{deleted}, & \text{if } (z''_A - z''_B) < 0 \\ \textit{unmodified}, & \text{otherwise.} \end{cases} \tag{5}$$

If the subtraction equals to zero, no reliable geometric difference can be detected using this metric alone. Thus, it is marked as *unmodified* and further differenced using other metrics below.

4.2 Color Difference

For practical purposes, color difference using Euclidean distance in RGBA space can provide little utility to the user. For instance in Fig. 6, a modeler might be concerned about the roof changing color from dark green to dark pink, yet they might not care about subtle changes from lighter to darker sand color in



(a) (b)

Figure 4: Depth differencing. Linearized depth values in (a) are subtracted from (b). The results are highlighted in red denoting deletion and in green denoting addition.

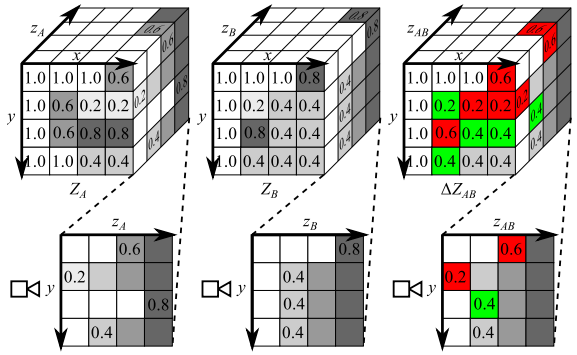


Figure 5: Linearized z-buffer differencing as a unit cube on a 4×4 screen. Top row: Depth changes in Z_A and Z_B are calculated in (x, y) screen space and visualized in ΔZ_{AB} as green for additions and red for deletions. Bottom row: Cross sections of the cube show the relative z-distances from camera.

the corner stones. This becomes increasingly important in scenes with too many color variations for the user to comprehend. For ease of argument we begin by considering differencing in the diffuse color only, as other material parameters can be constructed in a similar way. Alpha transparency also has to be treated separately due to blending which makes it view dependent.

Firstly, effects that are due to lighting need to be removed to suppress influence of any extrinsic factors. This avoids banding artefacts that may be caused by fragment interpolation over identical geometry but with different triangulation. Thus, the input here are two pre-aligned, unlit, unshaded and untextured diffuse color buffers for A and B respectively. Unlike standard color spaces such as RGB, HSV and HSL, the CIELAB [2008] and CIELUV [2009] attempt to provide improved perceptual uniformity across the tonal range. Hence, to take into account the perceptual differences between colors c_A and c_B, we base ΔC_{AB} on the original CIELAB color-difference [Robertson 1977] as

$$\Delta E^*_{AB} = \left\| (L^*_A, a^*_A, b^*_A) - (L^*_B, a^*_B, b^*_B) \right\|, \tag{6}$$

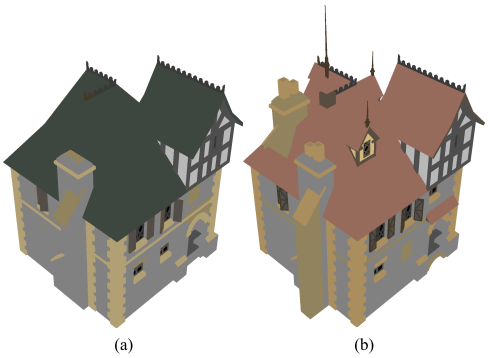


Figure 6: Diffuse color differencing. Unlit, unshaded and untextured diffuse color values are subtracted between two frames. Their differences are displayed in (a) and (b) in original coloring while unmodified parts are in greyscale.

where L^* corresponds to the lightness of the color, a^* its position between the red and green, and b^* between the yellow and blue values in the L*a*b* color space. Based on Eq. (2), for $\Delta C_{AB} = \Delta E^*_{AB}$, the value of $\epsilon \approx 2.3$ is in experimental psychology known as *just-noticeable difference (JND)* [Sharma and Bala 2002]. Color differences below this threshold are effectively imperceivable by the human visual system, hence labelled as *unmodified*, although in practice, the threshold could be set by the user using a sliding scale. Similarly, it is possible to visualize the 'rate of change' as a heat map of ΔC_{AB}.

4.3 Texture Difference

Texture difference can occur due to changes in the source image, its mapping onto the surface or both. However, simultaneous changes in both can cancel each other out. For instance, shifting a repeated pattern image as well as its mapping coordinates by exactly half a period each would result in the 3D rendering looking unmodified. Yet, the user might want to get notified of such a modification despite the lack of apparent visual discrepancy. Thus, texture maps need to be compared for changes in both color and parameter space, same as normal maps further discussed in §4.4.

Changes in fragment color due to the application of a texture can be detected similarly to color differencing in §4.2. Texel values (t_A, t_B) have to be sampled uncolored, unlit and unshaded as shown in Fig. 7. There, it is important to disregard all material properties as those, depending on the selected blending parameters, would attenuate the rendering and therefore skew the differencing. However, to apply texturing on a 3D model, each side of a source image needs to be parametrized in a chosen coordinate space P. For ease of argument we consider only (u, v) space as change detection in other spaces can be constructed in a similar way. Unlike previous metrics, texture parameters are not influenced by scene appearance due to lighting, shading, etc. They are also not view dependent and do not require normalization. Hence, the differences in p_A and p_B parameters can be easily determined as

$$\Delta P_{AB} = \| (u_A, v_A) - (u_B, v_B) \|, \tag{7}$$

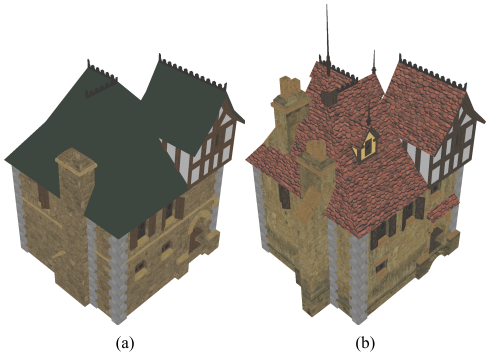


Figure 7: Texel differencing. Unlit, unshaded and uncolored but fully textured values in (a) are subtracted from those in (b). The differences are left in their original coloring while the rest of the scene is suppressed in greyscale.

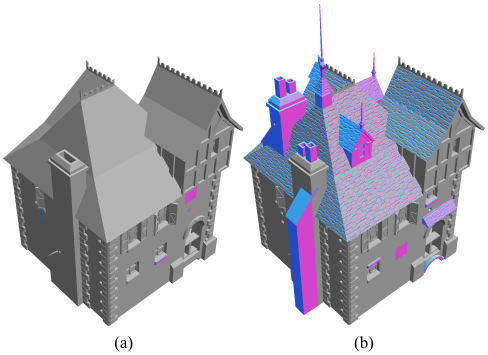


Figure 8: Normal differencing. Per-fragment input normals in (a) and (b) are differenced with unmodified sections in greyscale. Notice the fine detail in the roof tiles being correctly distinguished from the unchanged surface normals.

where u and v are the texture coordinates in range $[0, 1]$. For texture differencing, the ϵ threshold is especially important as depending on the image resolution and the mapping function, small fluctuations in the parameter vector might fall within the same texel.

4.4 Normal Difference

Even though normal maps could be directly compared using the underlying source images, it would exclude the effect that the geometry has on transforming the normals into a local tangent space as shown in Fig. 8. Thus, it is important to identify all changes in normal orientation mapped across the visible surfaces as well as the mapping parameters already discussed in §4.3.

For dot product, the projection of a unit normal $\hat{n}$ onto itself does not change its length, hence $\hat{n}_A \cdot \hat{n}_B = 1$. Conversely, the dot product of a normal with its inverted self yields -1. Distinguishing the latter case is generally useful as inverted normals are a frequent cause of lighting artefacts in 3D rendering. Based on this, a semantic meaning can be assigned to all $(\hat{n}_A, \hat{n}_B)$ as

$$\Delta N_{AB} := \begin{cases} added, & \text{if } \hat{n}_A = \vec{0} \wedge \hat{n}_B \neq \vec{0} \\ deleted, & \text{if } \hat{n}_A \neq \vec{0} \wedge \hat{n}_B = \vec{0} \\ inverted, & \text{if } (\hat{n}_A \cdot \hat{n}_B) = -1 \\ modified, & \text{if } (\hat{n}_A \cdot \hat{n}_B) \neq 1 \\ unmodified, & \text{otherwise,} \end{cases} \tag{8}$$

where $\vec{0}$ signifies an undefined or zero vector.

4.5 Metadata Difference

Several 3D formats including COLLADA, FBX and IFC contain metadata describing non-rendering specific properties of the geometry. Popular AEC tools such as Solibri Model Checker and Autodesk Dynamo let users create custom rule sets to further evaluate and visualize changes in metadata. In a similar way, we aim to detect all areas where metadata changed between revisions.

The original *Diff* algorithm by Hunt [1975] finds a common subsequence, i.e. an edit distance, of two text files on a line-by-line basis. In Screen Space 3D Diff we attempt to find those metadata that have changed on a fragment-by-fragment basis instead. Instead of exhaustive comparison, we can calculate approximate differences by encoding the data in a separate render pass. This neatly falls into the same metric differencing methodology as before. Encoding can be performed using any hashing function that transforms metadata into values within a 32-bit rendering buffer, e.g. Adler-32 [1996]. Once the modified metadata is identified, a standard line-by-line text differencing can be further performed on demand.

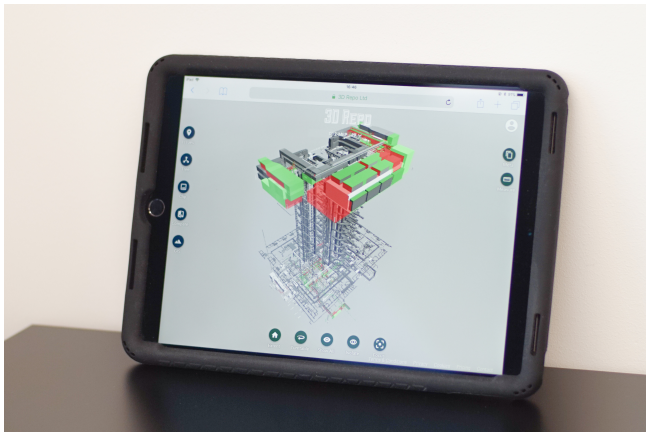


Figure 9: Tablet support. Our Unity-based implementation available at https://www.3drepo.io runs on desktops and tablets alike. Model courtesy of Canary Wharf Contractors.

5 IMPLEMENTATION

Screen Space 3D Diff has been implemented in both *3D Repo GUI* C++ desktop application and *3drepo.io* web client. The latter is built using the Unity game engine , c.f. [Friston et al. 2017], which relies on a component based programming model. This functionality is implemented in highly specialized Components (objects) that are attached to GameObjects, i.e. scene graph nodes, and invoked via callbacks. Our Unity application, therefore, consists of two parts; i) A Component generated texture buffers containing the operands from the model data, and ii) a post-processing effect which implements the difference metric and resulting visualization.

Models. Both A and B models are loaded into the same scene graph but under different branches. The scenes are already pre-aligned as they come from the same project stored in 3D Repo. Unity has the ability to assign GameObjects to 'Layers' that can be used to cull meshes attached to GameObjects during rendering. B-branch GameObjects are, therefore, assigned to one such layer, which is culled by the main color camera in the default configuration.

Operand rendering. A dedicated '3D Diff' Component produces the operand buffers. It uses Unity's Replacement Shader functionality to render all scene geometry with a specific shader. The camera renders to a single-precision depth texture. The shader itself uses the default Unity depth functionality: it writes the inverse reciprocal of the view-space depth. Two passes are required, one culling everything but the A model, and one culling everything but the B model. Each pass then writes to a separate texture. Camera culling

masks can also be used by the 3D Diff Component to implement different viewing configurations, see Fig. 1.

5.1 Visualization

Unity's post-processing Effect functionality is used to display the results. This is invoked after the main color pass. Each Effect copies the color pass from one buffer to another by utilizing a full-screen quad rendered with a typical shader that reads the color buffer as a texture. This way, screen space effects can be applied to the color pass, one fragment shader invocation per pixel. The final effect renders to the front buffer. As Effects are traditional shaders, they can utilize a number of resources. The 3D Diff Effect binds the aforementioned depth textures and implements the difference metric. In the fragment shader, the color is sampled and converted to greyscale. The depth values are sampled and compared, and a red or green hue applied if their difference exceeds a constant threshold. Other visualizations can be easily implemented, however. The 3D Diff tool does not explicitly have to render the scene as the Effect operates on the main Unity color pass. This means the tool can not only overlay different combinations of the *A* and *B* models as above, but overlays the effects of other arbitrary components without needing any modifications to the Effect pass.

Depth precision. Unity uses a reversed-z depth buffer which better distributes the precision [Lapidous and Jiao 1999]. AEC models are larger than typical game assets, so even the inverse distribution can be insufficient. In this case, logarithmic depth buffers can be used. This is done by explicitly writing the depth in the Replacement Shader. The vertex shader computes the view space depth and passes it as a single-precision fragment parameter. The fragment shader then encodes the depth using a logarithmic function and writes it to the depth texture. The inverse is performed in the 3D Diff Effect shader to recover the original view-space depth for the difference metric. Regardless of the storage format, the rasterizer will always work with the reciprocal of the depth needed for correct interpolation under perspective projection. This not only limits the precision but makes the metric threshold view and clip plane dependent. An enhanced implementation could perform a ray-plane intersection test in the fragment shader, with plane parameters from a geometry shader to compute the absolute per-pixel eye space depth. This could be stored in a floating point or RGBA texture. The hardware depth testing would still use the reciprocal, requiring a logarithmic buffer or multiple passes to avoid rendering artefacts. However, it would allow truly linear depth, and also enable single-precision storage on platforms that do not currently support it.

6 EVALUATION

The proposed method has been evaluated on a number of industrial 3D models with several engineering practices. Special attention was put on encompassing a variety of authoring tools and modeling techniques but also on demonstrating the rendering being independent of scene complexity. We ensured that the method produced consistent visualization without flickering, misalignments or other visual artefacts due to quantization errors. In addition, its usability was evaluated in a user study further described in §6.2.

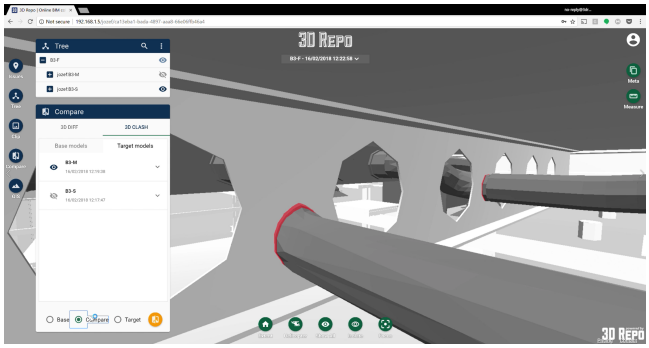


Figure 10: Clash detection. Overlapping surfaces can be detected as an inverse of differencing. Here, mechanical (Fig. 1) and structural (Fig. 9) models of the same building are examined for design clashes which are highlighted in red.

6.1 Results

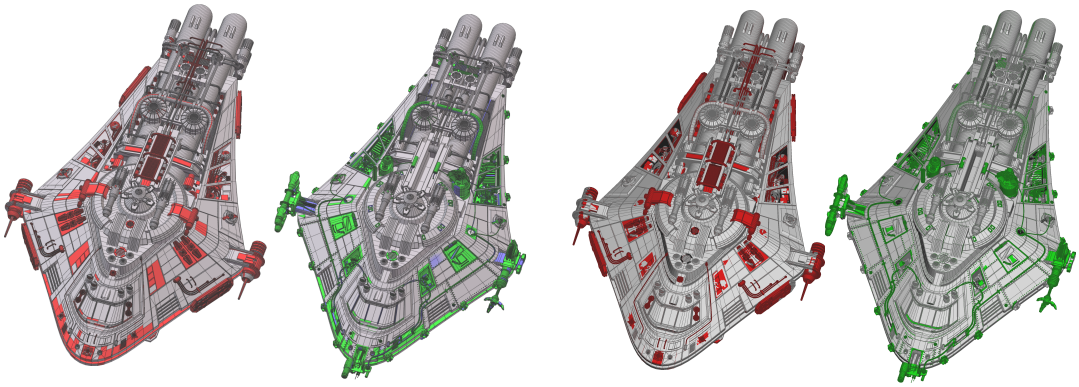
Our solution is independent of any specific rendering technique and works equally well on points, lines and polygons. Fig. 3 (c) and (d) shows that object-level changes and changes in the underlying triangulation are correctly identified and highlighted. Similarly, Fig. 1 and Fig. 9 show differences in engineering models of a multi-storey building. Despite a large number of components and polygons, our method operates in real-time with minimal overhead. Tab. 1 lists performance using various models across different devices and screen resolutions. In addition, an inverse of the differencing calculation highlights not the changes but rather commonalities, i.e. areas where the models overlap, known as *clash detection* in the AEC industry, see Fig. 10. The final result is presented in Fig. 11 where our method is compared with MeshGit [2013]. The intention was to emulate the visualization of MeshGit using their largest 3D model in order to demonstrate that screen space differencing can produce directly comparable results at a fraction of the computational cost.

6.2 User Study

We also conducted a preliminary user study with 20 CAD/BIM professionals from some of the largest architectural and construction companies in the UK. A balanced mix of beginner (6), intermediate (7) and expert (7) users working daily in tools such as Autodesk Revit, Navisworks and Bentley Microstation were recruited. In order to contrast the *3D Diff* interface, we also developed a *Sync View* visualization to provide a basic side-by-side rendering with interlinked navigation for manual comparison of 3D models. Before

[FPS]	**Thinkpad Yoga 260** Intel HD Graphics 520 (1920x1080px)	**iPad Pro 10"** A10X (2224x1668px)	**MacBook Pro 15"** Radeon Pro 560 (2880x1800px)	**Cyberpower PC** Nvidia GeForce Titan (4096x2160px)
Medieval	60	60	60	60
Shuttle	55	60	60	60
Mechanical	16	20	60	60

Table 1: Frames per second measurements across various devices and screen resolutions using sample models: Medieval (Fig. 3); Shuttle (Fig. 11); and Mechanical (Fig. 9). Most are capped at V-Sync rate of the screen at hand.



(a) MeshGit (b) Screen Space 3D Diff

Figure 11: Comparison with MeshGit [2013] on their Shuttle model with over 3k components and 190k polygons. On comparable hardware, MeshGit in (a) takes 9.7 minutes while our method in (b) is real-time. Notice how our method segregates surface-level changes from modifications in the underlying objects, the geometry and vertex-level connectivity shown in Fig. 3.

each trial, participants were given a sample dataset to familiarize themselves with the given interface and then a pair of architectural 3D models to detect geometric additions and deletions. After each session, users completed a system usability scale (SUS) [1996] questionnaire. The order of the interfaces and datasets was shuffled according to Latin square. Tab. 2 lists median values for both time to completion and SUS scores. 3D Diff reached grade B, i.e. an above the average user interface, while Sync View only grade C, i.e. a below an average user interface. Accordingly, the users found 3D Diff easier to use and were more confident in its functionality. Overall, the users agreed 3D Diff would provide a quick and easy way of determining changes in day-to-day engineering tasks. Nevertheless, two users preferred the Sync View interface due to a larger field of view. Many users independently requested to export a written report of detected object IDs similar to clash detection in AEC. Also, a few requested to turn geometry transparent what is a limitation.

		Sync View		3D Diff	
	Experience	Time [m]	SUS	Time [m]	SUS
P1	Expert	14	82.5	15	85.0
P2	Intermediate	14	80.0	22	90.0
P3	Expert	10	92.5	10	85.0
P4	Intermediate	13	72.5	10	80.0
P5	Beginner	8	80.0	22	75.0
P6	Intermediate	30	42.5	27	70.0
P7	Intermediate	37	50.0	36	50.0
P8	Expert	14	92.5	13	95.0
P9	Beginner	14	90.0	65	75.0
P10	Expert	22	67.5	17	87.5
P11	Intermediate	29	40.0	17	60.0
P12	Intermediate	5	62.5	10	65.0
P13	Expert	27	47.5	8	50.0
P14	Beginner	20	37.5	20	55.0
P15	Expert	25	62.5	23	62.5
P16	Beginner	11	90.0	24	85.0
P17	Expert	17	47.5	10	42.5
P18	Intermediate	29	32.5	5	62.5
P19	Beginner	20	20.0	32	60.0
P20	Beginner	22	75.0	8	100.0
Median	Intermediate	18	**65.0**	17	**72.5**

Table 2: User study results in Sync View vs 3D Diff based on time to completion and system usability scale.

7 DISCUSSION

As demonstrated in §6, by calculating differences in the image space, we avoid the need for computationally expensive correspondence estimation, such as in MeshGit [2013] or 3D Timeline [2014], each of which might still lead to errors and omissions. In essence, our pre-alignment of the input models and their subsequent comparison based on the same screen space co-locations defines a quasi-correspondence that is sufficient for our purposes. Nevertheless, it is important to note that this approach does not replace actual object-level differencing in the world space. Our method is merely a faster approach to get a fragment-based differencing in real-time. One of the main insights is that in order to present meaningful results, it is not necessary to calculate all changes in the entire scene. Instead, it is sufficient to find only those changes that are currently visible. Since this is a screen space method, its complexity is $O(n)$ per metric, where n is the number of rendered pixels.

Limitations. Despite obvious advantages such as real-time performance, ease of implementation and massive scalability, it still suffers from several limitations. Fig. 1 shows that simply repositioning an object results in it being highlighted as deletion in A and addition in B. This is because it has no notion of object-level correspondence. Nonetheless, as demonstrated in our user study in §6.2, professionals still value these results as they highlight changes in complex scenes. Although in general any unrelated models can be compared using our method, the most meaningful results are produced from revisions of the same scene. Unfortunately, overlapping surfaces of different objects in Fig. 1 register as unmodified. This is technically correct as based on what is visible, those surfaces indeed occupy the same space in world coordinates. Another limitation is that it cannot reveal changes on occluded objects.

8 CONCLUSIONS

The process of 3D rendering encompasses many disparate layers that together produce the final image in the screen space. These include not only the geometry but also materials, lights, normals, textures, environmental effects and so on. In order to achieve meaningful results when differencing 3D scenes, each of these constituents needs to be isolated and compared individually. Therefore, in this paper, we presented Screen Space 3D Diff, a novel method for fast and reliable discovery of multi-metric changes between two input models suitable for a web-based implementation. Unlike existing solutions, our method does not require any computationally expensive correspondence estimation and is independent of both scene complexity and data representations. This makes it extremely suitable for real-time implementation in the Unity game engine. Due to these crucial characteristics, it presents a uniquely compelling solution for the AEC industry where there are often very complex scenes generated by a myriad of modeling tools and processes. One of the limitations of Screen Space 3D Diff is its inability to detect differences in occluded areas which is left for future work.

ACKNOWLEDGMENTS

We would like to thank the participants of our user study from Arup, Balfour Beatty, Canary Wharf Contractors and Crossrail for their help and feedback. This work has been supported by Innovate UK Infrastructure Systems grant No. 102813 as well as Horizon 2020 grant No. 700294. Patent applied for: PCT/EP2018/052006.

REFERENCES

Dror Aiger, Niloy J. Mitra, and Daniel Cohen-Or. 2008. 4-points Congruent Sets for Robust Pairwise Surface Registration. *ACM Trans. Graph.* 27, 3, Article 85 (Aug. 2008), 10 pages. https://doi.org/10.1145/1360612.1360684

Johannes Behr, Peter Eschler, Yvonne Jung, and Michael Zöllner. 2009. X3DOM: A DOM-based HTML5/X3D Integration Model. In *Proceedings of the 14th International Conference on 3D Web Technology (Web3D '09)*. ACM, New York, NY, USA, 127–135. https://doi.org/10.1145/1559764.1559784

John Brooke et al. 1996. SUS-A quick and dirty usability scale. *Usability evaluation in industry* 189, 194 (1996), 4–7.

Hsiang-Ting Chen, Li-Yi Wei, and Chun-Fa Chang. 2011. Nonlinear Revision Control for Images. *ACM Trans. Graph.* 30, 4, Article 105 (July 2011), 10 pages. https://doi.org/10.1145/2010324.1965000

Jonathan D. Denning, William B. Kerr, and Fabio Pellacini. 2011. MeshFlow: Interactive Visualization of Mesh Construction Sequences. *ACM Trans. Graph.* 30, 4, Article 66 (July 2011), 8 pages. https://doi.org/10.1145/2010324.1964961

Jonathan D. Denning and Fabio Pellacini. 2013. MeshGit: Diffing and Merging Meshes for Polygonal Modeling. *ACM Trans. Graph.* 32, 4, Article 35 (July 2013), 10 pages. https://doi.org/10.1145/2461912.2461942

Peter Deutsch and Jean-Loup Gailly. 1996. *Zlib compressed data format specification version 3.3.* Technical Report.

Jozef Doboš, Niloy J. Mitra, and Anthony Steed. 2014. 3D Timeline: Reverse Engineering of a Part-based Provenance from Consecutive 3D Models. *Comput. Graph. Forum* 33, 2 (May 2014), 135–144. https://doi.org/10.1111/cgf.12311

Jozef Doboš, Kristian Sons, Dmitri Rubinstein, Philipp Slusallek, and Anthony Steed. 2013. XML3DRepo: A REST API for Version Controlled 3D Assets on the Web. In *Proceedings of the 18th International Conference on 3D Web Technology (Web3D '13)*. ACM, NY, USA, 47–55. https://doi.org/10.1145/2466533.2466537

Jozef Doboš and Anthony Steed. 2012a. 3D Diff: an interactive approach to mesh differencing and conflict resolution. In *SIGGRAPH Asia 2012 Technical Briefs (SA '12)*. ACM, NY, USA, Article 20, 4 pages. https://doi.org/10.1145/2407746.2407766

Jozef Doboš and Anthony Steed. 2012b. 3D revision control framework. In *Proceedings of the 17th International Conference on 3D Web Technology (Web3D '12)*. ACM, New York, NY, USA, 121–129. https://doi.org/10.1145/2338714.2338736

Sebastian Friston, Carmen Fan, Jozef Doboš, Timothy Scully, and Anthony Steed. 2017. 3DRepo4Unity: Dynamic Loading of Version Controlled 3D Assets into the Unity Game Engine. In *Proceedings of the 22Nd International Conference on 3D Web Technology (Web3D '17)*. ACM, New York, NY, USA, Article 15, 9 pages. https://doi.org/10.1145/3055624.3075941

Z. Han, Z. Liu, C. M. Vong, Y. S. Liu, S. Bu, J. Han, and C. L. P. Chen. 2017. BoSCC: Bag of Spatial Context Correlations for Spatially Enhanced 3D Shape Representation. *IEEE Transactions on Image Processing* 26, 8 (Aug 2017), 3707–3720. https://doi.org/10.1109/TIP.2017.2704426

J. W. Hunt and M. D. McIlroy. 1975. *An algorithm for differential file comparison. Computer Science.* Technical Report.

ISO/TC 184/SC 4. 1994. *Industrial automation systems and integration – Product data representation and exchange.* Technical Specification. International Organization for Standardization. ISO 10303-1:1994.

ISO/TC 184/SC 4. 2012. *Industrial automation systems and integration – COLLADA digital asset schema specification for 3D visualization of industrial data.* Technical Specification. Khronos Group. ISO/PAS 17506:2012.

ISO/TC 184/SC 4. 2013. *Industry Foundation Classes (IFC) for data sharing in the construction and facility management industries.* Technical Specification. International Organization for Standardization. ISO 16739:2013.

ISO/TC 274. 2008. *Colorimetry - Part 4: CIE 1976 L*a*b* Colour space.* Technical Specification. Commission internationale de l'ï£¡clairage. ISO 11664-4:2008.

ISO/TC 274. 2009. *Colorimetry - Part 5: CIE 1976 L*u*v* Colour space and u', v' uniform chromaticity scale diagram.* Technical Specification. Commission internationale de l'ï£¡clairage. ISO 11664-5:2009.

Arjun Jain, Thorsten Thormählen, Tobias Ritschel, and Hans-Peter Seidel. 2012. Exploring Shape Variations by 3D-Model Decomposition and Part-based Recombination. *Comp. Graph. Forum (Proc. Eurographics 2012)* 31, 2 (2012).

Svenja Kahn. 2013. Reducing the Gap Between Augmented Reality and 3D Modeling with Real-time Depth Imaging. *Virtual Real.* 17, 2 (June 2013), 111–123. https://doi.org/10.1007/s10055-011-0203-0

Maja Krivokuća, Burkhard C. Wünsche, and Waleed Abdulla. 2012. A New Error Metric for Geometric Shape Distortion Using Depth Values from Orthographic Projections. In *Proceedings of the 27th Conference on Image and Vision Computing New Zealand (IVCNZ '12)*. ACM, 6. https://doi.org/10.1145/2425836.2425911

In Kyu Park, Kyoung Lee Mu, and Sang Lee Uk. 2002. Efficient Measurement of Shape Dissimilarity between 3D Models Using Z-Buffer and Surface Roving Method. 2002 (10 2002).

Eugene Lapidous and Guofang Jiao. 1999. Optimal depth buffer for low-cost graphics hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*. ACM, 67–73. https://doi.org/10.1145/311534.311579

Martin Mittring. 2007. Finding Next Gen: CryEngine 2. In *ACM SIGGRAPH 2007 Courses (SIGGRAPH '07)*. ACM, New York, NY, USA, 97–121. https://doi.org/10.1145/1281500.1281671

Alan R Robertson. 1977. The CIE 1976 Color-Difference Formulae. *Color Research & Application* 2, 1 (1977), 7–11.

Timothy Scully, Jozef Doboš, Timo Sturm, and Yvonne Jung. 2015. 3Drepo.Io: Building the Next Generation Web3D Repository with AngularJS and X3DOM. In *Proceedings of the 20th International Conference on 3D Web Technology (Web3D '15)*. ACM, New York, NY, USA, 235–243. https://doi.org/10.1145/2775292.2775312

Timothy Scully, Sebastian Friston, Carmen Fan, Jozef Doboš, and Anthony Steed. 2016. glTF Streaming from 3D Repo to X3DOM. In *Proceedings of the 21st International Conference on Web3D Technology (Web3D '16)*. ACM, New York, NY, USA, 7–15. https://doi.org/10.1145/2945292.2945297

Perumaal Shanmugam and Okan Arikan. 2007. Hardware Accelerated Ambient Occlusion Techniques on GPUs. In *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games (I3D '07)*. ACM, New York, NY, USA, 73–80. https://doi.org/10.1145/1230100.1230113

Gaurav Sharma and Raja Bala. 2002. *Digital color imaging handbook.* CRC press. ISBN 0-8493-0900-X.

Kristian Sons, Felix Klein, Dmitri Rubinstein, Sergiy Byelozyorov, and Philipp Slusallek. 2010. XML3D: Interactive 3D Graphics for the Web. In *Proceedings of the 15th International Conference on Web 3D Technology (Web3D '10)*. ACM, NY, USA, 175–184. https://doi.org/10.1145/1836049.1836076