Architectural Paradigms for 2.5D Neural Relighting:
A Comprehensive Technical Analysis of Web-Based Implementation

Research Report

December 26, 2025

1 Introduction: The Renaissance of Intrinsic Image Decomposition

The manipulation of illumination within digital imagery has historically been the province of high-end post-production suites and computational photography pipelines restricted to desktop environments. Software such as DaVinci Resolve has popularized the concept of "relighting"—the ability to synthetically alter the lighting conditions of a scene after capture. This capability fundamentally relies on the reconstruction of three-dimensional geometry from two-dimensional inputs, a process known in computer vision as intrinsic image decomposition. By isolating the geometry (depth and normals) from the albedo (base color), visual effects artists can introduce virtual light sources that interact physically with the subject, casting realistic shadows and generating specular highlights that respect the scene's topography.

With the advent of WebGPU and highly optimized client-side inference engines like ONNX Runtime Web, this capability is now migrating from native applications to the web browser. The convergence of Monocular Depth Estimation (MDE) foundation models—specifically **Depth Anything V2**—and hardware-accelerated graphics pipelines allows developers to build sophisticated photo editing tools that operate entirely on the client's device.

This report provides an exhaustive analysis of the theoretical and practical frameworks required to implement a robust, browser-based relighting tool. It evaluates the current state-of-the-art (SOTA) in depth estimation, explores the mathematics of surface reconstruction and screen-space raymarching for shadow generation, and details the engineering architecture necessary to achieve real-time performance through zero-copy GPU interoperability.

2 Theoretical Foundations of Single-Image Relighting

To replicate the fidelity of professional grading tools, one must move beyond simple 2D brightness adjustments and adopt a "2.5D" rendering approach. In this paradigm, the image is treated not as a flat array of pixels but as a textured surface with varying elevation. The realism of the resulting lighting effect depends on the accuracy of three critical components: the Depth Map, the Surface Normal Map, and the Shadow/Occlusion Model.

2.1 The Depth Map as Topographical Displacement

At the core of the relighting engine is the depth map—a single-channel image where pixel intensity corresponds to the distance from the camera's focal plane. In a web-based relighting context, the depth map functions as a displacement field. It provides the spatial segregation necessary to illuminate a foreground subject independently of the background, mimicking the behavior of physical stage lighting.

The precision of this map is paramount. If the depth estimation fails to capture fine details—such as strands of hair or the geometric edge of a table—the subsequent relighting will appear "detached," creating a halo effect where the light does not perfectly track the object's contours. Modern workflows utilize these maps to define "spatial zones" for targeted color grading, allowing for volumetric adjustments that were previously impossible in 2D compositing.

2.2 Surface Normals and Lambertian Reflectance

While depth determines *where* an object is, surface normals determine *how* it reflects light. A depth map alone cannot produce realistic shading gradients; for that, the rendering engine requires a Normal Map. A normal vector represents the perpendicular direction of the surface at any given pixel. In 2.5D relighting, these normals are rarely captured by sensors but are instead mathematically derived from the depth map.

The interaction between the virtual light and the image surface is governed by lighting models such as Lambertian Reflectance or Physically Based Rendering (PBR) approximations. The fundamental calculation relies on the dot product between the surface normal (N) and the light direction vector (L).

$$I = \text{Albedo} \times \max(N \cdot L, 0)$$

This equation dictates that surfaces facing the light source receive maximum illumination, while those angling away fall into darkness. DaVinci Resolve's "Relight FX" utilizes this principle, generating a surface map that reflects light following the curvature of the subject. Without accurate normal reconstruction, the lighting appears flat and artificial, lacking the nuanced falloff seen on curved surfaces like faces or cylindrical objects.

2.3 The Physics of Screen-Space Shadows

Perhaps the most challenging aspect of single-image relighting is the generation of cast shadows. Unlike shading, which is a local property of the surface angle, shadows are global phenomena caused by occlusion. In a full 3D environment, shadows are calculated using shadow maps or ray tracing against a mesh. In 2.5D relighting, we lack a complete mesh and must rely on **Screen-Space Raymarching**.

Raymarching involves tracing a vector from the pixel's position towards the virtual light source. As the ray traverses the screen space, the algorithm samples the depth map at each step. If the ray's depth (Z_{ray}) becomes greater than the scene's depth (Z_{scene}) at that coordinate, it indicates that an object stands between the pixel and the light, casting a shadow.

The implementation of "realistic shadows and all," as requested, requires sophisticated handling of this raymarching process to avoid artifacts such as "shadow acne" (self-shadowing due to precision errors) and "peter-panning" (detached shadows). Furthermore, creating soft shadows—where the shadow blurs as it gets further from the caster—requires accumulating penumbra values during the march, mimicking the behavior of area lights rather than point lights.

3 Evaluation of Depth Estimation Models

The efficacy of a web-based relighting tool is entirely dependent on the quality of the underlying depth estimation model. The years 2024 and 2025 have witnessed a divergence in model architectures, splitting primarily into **Discriminative** (Transformer-based) and **Generative** (Diffusion-based) approaches.

3.1 Comparative Analysis of SOTA Models

The following analysis evaluates the leading contenders—**Depth Anything V2**, **Marigold**, and **ZoeDepth**—based on criteria critical for web deployment: inference latency, edge precision, and robustness to "in-the-wild" imagery.

Metric	Depth Anything V2	Marigold	ZoeDepth	MiDaS v3.1
Architecture	Discriminative (ViT/DINOv2)	Generative (Stable Diff.)	Discriminative (ViT/CNN)	Discriminative (ViT)
Inference Speed	**Real-time (<50ms)**	Offline (Sec/Min)	Moderate	Moderate
Detail Preservation	**High** (Fine edges, hair)	**Ultra-High** (Hallucinated)	Moderate	Low (Blobby edges)
Robustness	Excellent	Variable	Good	Average
Web Suitability	**Optimal**	Poor	Moderate	High
Training Data	62M+ Real + Synthetic	Synthetic Latent Space	NYU/KITTI (Specific)	General Mix

Table 1: Comparative Analysis of Depth Estimation Models

3.1.1 Depth Anything V2: The Discriminative Foundation

Released in mid-2024, **Depth Anything V2** has established itself as the premier foundation model for monocular depth estimation. Unlike its predecessors, it is trained on a massive corpus of 62 million unlabeled real images and nearly 600,000 synthetic images with ground truth. This hybrid training regime allows it to generalize exceptionally well across diverse scenarios—from close-up portraits to complex outdoor landscapes.

- **Key Advantage for Web:** The model is highly efficient. It is approximately **10 times faster** than diffusion-based alternatives like Marigold. It is available in scalable sizes, with the "Small" (24.8M parameters) and "Base" (97.5M parameters) variants being perfectly sized for client-side execution within a browser environment.
- **Performance:** It significantly outperforms ZoeDepth and MiDaS in zero-shot relative depth estimation, particularly in preserving high-frequency details. This is crucial for relighting, as accurate edges prevent light "leaking" from the foreground to the background.

3.1.2 Marigold: The Generative Approach

Marigold represents a different philosophy, repurposing the latent space of Stable Diffusion to "denoise" depth maps. By leveraging the semantic knowledge embedded in a text-to-image model, Marigold can infer depth in highly ambiguous regions (e.g., distinguishing a painting on a wall from a window).

- **Detail vs. Speed:** While Marigold produces sharper, more detailed maps than any discriminative model, it comes at a massive computational cost. A single inference can take several seconds to minutes on a consumer GPU, making it unsuitable for an interactive "real-time" photo editing tool where users expect instant feedback when adjusting light sliders.

3.1.3 ZoeDepth: The Metric Specialist

ZoeDepth was previously the standard for metric depth estimation (predicting absolute distance in meters).

- **Limitation:** While accurate for measurement, ZoeDepth often produces "coarser" maps compared to Depth Anything V2. In relighting, *relative* depth accuracy (knowing distinct layers of the image) and edge sharpness are more valuable than absolute metric precision. Depth Anything V2 has effectively superseded ZoeDepth in robust generalization.

3.2 Strategic Recommendation for Tool Development

For the specific purpose of a web-based photo editing tool, **Depth Anything V2 (Small variant)** is the recommended engine.

1. **Latency:** The Small model's lightweight architecture allows for near-instantaneous inference on modern GPUs via WebGPU, enabling a responsive user experience.
2. **Edge Fidelity:** Its ability to resolve fine structures (like hair or vegetation) ensures that generated normal maps are crisp, resulting in high-quality lighting interactions.
3. **Commercial Viability:** The Apache 2.0 license and open ONNX availability make it a safe and practical choice for tool development.

4 Technical Architecture: The "Zero-Copy" Web Pipeline

Implementing this tool in a browser requires overcoming the traditional bottleneck of web-based AI: the transfer of data between the CPU (JavaScript) and the GPU. A naive implementation—downloading the AI output to the CPU to create a texture—destroys performance. The architecture proposed here utilizes **WebGPU** and **ONNX Runtime Web** to create a "Zero-Copy" pipeline where data resides permanently on the GPU.

4.1 The Stack Components

- **Inference Engine: ONNX Runtime Web (ORT-Web)** with the `webgpu` execution provider. This allows the neural network to execute purely on the GPU.
- **Graphics API: WebGPU**, accessed via **Three.js**. WebGPU offers compute shaders and storage textures that are essential for handling the output of the neural network efficiently.
- **Rendering Engine: Three.js** with the new `WebGPURenderer` and **TSL (Three Shading Language)**. TSL allows for the creation of node-based materials that can directly consume WebGPU buffers.

4.2 The Data Flow Pipeline

The critical innovation in this architecture is **IO Binding**.

1. **Texture Upload:** The user's image is uploaded to a WebGPU texture (`GPUTexture`).
2. **Tensor Creation (Zero-Copy):** Instead of converting the image to a CPU array, we create an ONNX tensor directly from the `GPUTexture`. This tells the inference engine to read the input data already present in VRAM.

3. **Inference:** The Depth Anything V2 model executes on the GPU.
4. **Output Binding (Zero-Copy):** The model writes its output (the depth map) directly into a pre-allocated `GPUBuffer` or `GPUTexture`. We explicitly instruct ONNX Runtime to keep the output on the GPU via `preferredOutputLocation: 'gpu-buffer'`.
5. **Rendering Interop:** The Three.js `WebGPURenderer` wraps this external `GPUBuffer` using `THREE.ExternalTexture` or `THREE.StorageTexture`. This allows the rendering shader to access the depth data immediately as a texture map without any CPU round-trip.

This pipeline ensures that high-resolution images (e.g., 2K or 4K) can be processed and relit at interactive framerates (30-60 FPS), limited only by the GPU's compute capability rather than PCIe bus bandwidth.

5 Algorithmic Implementation: From Depth to Light

Once the depth map is resident in GPU memory, the relighting logic is handled by custom shaders. This section details the mathematical algorithms required to generate surface normals, shading, and shadows.

5.1 Step 1: Normal Map Reconstruction (The Derivative Approach)

The raw depth map provides scalar height data (h). To light the image, we need vector surface normals (n). These are derived by calculating the gradient of the depth map—the rate of change in height along the X and Y axes.

While sophisticated methods exist, the **Sobel Operator** or **Central Difference** method is most efficient for real-time web shaders. For a pixel at coordinates (u, v) with depth $D(u, v)$:

1. **Neighbor Sampling:** Sample the depth of adjacent pixels:
   - $z_{left} = D(u - \epsilon, v)$
   - $z_{right} = D(u + \epsilon, v)$
   - $z_{up} = D(u, v - \epsilon)$
   - $z_{down} = D(u, v + \epsilon)$
   - *Note: ϵ is the texel size ($1.0/$resolution).*
2. **Gradient Calculation:**
$$\frac{dz}{dx} = \frac{z_{right} - z_{left}}{2.0}$$
$$\frac{dz}{dy} = \frac{z_{down} - z_{up}}{2.0}$$
3. **Normal Construction:** The normal vector is the cross product of the tangent vectors. Simplified for screen space, it is:
$$N = \text{normalize}\left(\vec{3}\left(-\frac{dz}{dx} \cdot S, -\frac{dz}{dy} \cdot S, 1.0\right)\right)$$
   **S is a "strength" or "smoothness" factor. Increasing S exaggerates the relief effect; decreasing it flattens the surface.*

5.2 Step 2: Screen-Space Raymarching for Shadows

To achieve the "real shadows" requested, we implement a simplified raymarching algorithm often termed **2.5D Ray Tracing** or **Screen Space Shadows (SSS)**.

The Algorithm:

1. **Reconstruct World Position:** For the current pixel, reconstruct its 3D position (P_{origin}) using its UV coordinates and depth value.
2. **Light Vector:** Calculate the direction vector (R_{dir}) towards the virtual light source.
3. **The March:**
   - Move along R_{dir} in small increments (steps).
   - At each step i, calculate the current position $P_{sample} = P_{origin} + R_{dir} \times \text{step}_i$.
   - Project P_{sample} back into screen coordinates (UV_{sample}) to look up the "ground truth" depth stored in the depth map (Z_{map}).
   - **Occlusion Test:** Compare the ray's current depth (Z_{ray}) with the map's depth (Z_{map}).
     - If $Z_{ray} > Z_{map} + \text{bias}$, it means the ray has gone *behind* an object in the depth map. The light is blocked.
   - **Break:** If occlusion is found, the pixel is in shadow. Stop marching.

5.3 Step 3: Horizon-Based Ambient Occlusion (HBAO)

To add further realism ("and all"), implementing **Ambient Occlusion (AO)** is crucial. Shadows from raymarching handle direct light blocking, but AO handles the soft darkening in cracks and crevices where light struggles to penetrate. **HBAO** is the industry standard for this. It marches rays in multiple directions around a pixel to determine the "horizon angle"—how much of the sky is visible from that point. This adds weight and contact hardening to objects, preventing them from looking like they are floating.

6 Practical Implementation Guide

The following roadmap outlines the construction of the tool using the identified technologies.

6.1 Phase 1: Environment & Model Setup

1. **Initialize Project:** Set up a Vite project with `three` (v0.166+ for WebGPU support) and `onnxruntime-web`.
2. **Model Conversion:** Download the **Depth Anything V2 Small** model. Use the `dynamo.py` script from the official repo to export it to ONNX with FP16 precision.
3. **ORT Session:** Initialize the session in JavaScript with `executionProviders: ['webgpu']`.

6.2 Phase 2: The WebGPU Inference Loop

1. **Image Loading:** Load the user image into an `HTMLImageElement` and create a Three.js `Texture`.
2. **Texture to Tensor:** Use `ort.Tensor.fromGpuBuffer` to wrap the underlying GPU buffer of the texture.

3. **Inference:** Run `session.run()`.
4. **Tensor to Texture:** The output will be a tensor representing the depth map. Bind this output buffer to a Three.js `StorageTexture` or `ExternalTexture` for the renderer.

6.3 Phase 3: The Relighting Shader (TSL)

Create a `MeshBasicNodeMaterial` or a custom `NodeMaterial` in Three.js using TSL.

```
// Pseudo-code for TSL Relighting Implementation
import { texture, uv, float, vec3, normalize, dot, max, mix } from 'three/tsl';

// 1. Inputs
const baseTexture = texture(imageTexture);
const depthTextureNode = texture(depthOutputTexture); // From ONNX
const lightPos = uniform(new Vector3(1, 1, 1)); // Controlled by mouse

// 2. Normal Reconstruction (Sobel-like)
const normalNode = Fn(() => {
    const val = depthTextureNode.sample(uv());
    const valRight = depthTextureNode.sample(uv().add(vec2(epsilon, 0)));
    const valUp = depthTextureNode.sample(uv().add(vec2(0, epsilon)));

    const dx = valRight.sub(val);
    const dy = valUp.sub(val);

    return normalize(vec3(dx.negate().mul(strength),
                          dy.negate().mul(strength),
                          1.0));
});

// 3. Lighting Calculation (Lambertian)
const lightDir = normalize(lightPos.sub(positionWorld));
const diffuse = max(dot(normalNode(), lightDir), 0.0);

// 4. Raymarching Shadow (Simplified Concept)
const shadowFactor = RaymarchShadowNode(depthTextureNode, positionWorld, lightDir);

// 5. Composite
const finalColor = baseTexture.mul(diffuse.mul(shadowFactor));
```

7 Conclusions and Strategic Outlook

The ability to implement professional-grade relighting in a web browser is a testament to the rapid maturation of web graphics and AI standards. The architecture defined in this report—anchored by **Depth Anything V2** for geometry and **WebGPU** for rendering—offers the optimal balance of performance and fidelity.

Key Takeaways:

- **Depth is the Driver:** The quality of relighting is strictly limited by the depth model. Depth Anything V2 is currently the only viable choice for high-fidelity, real-time web deployment.

- **Zero-Copy is Mandatory:** For interactive applications, data must stay on the GPU. The CPU-GPU bottleneck is the primary killer of performance in web AI. Utilizing ONNX Runtime's IO Binding is not optional; it is an architectural requirement.
- **2.5D is an Illusion:** The entire effect relies on clever shader mathematics (raymarching, normal reconstruction) to trick the eye. Understanding the limits of this illusion (e.g., occlusion artifacts) is key to managing user expectations.

By following this blueprint, developers can construct a tool that not only matches the feature set of native applications like ClipDrop but does so with the privacy and accessibility inherent to the web platform. The future of photo editing is not just pixels; it is physics, simulated in real-time, in the palm of your hand.