ChatGPT

Raytracing via Emissive Surfaces in Babylon.js

This guide shows how to **fake ray-tracing** by treating bright surfaces as light emitters and using Babylon.js features (emissive materials, post-processes, probes, etc.) to approximate reflections, bounce light, soft shadows, and global illumination. We'll outline key APIs (emissive color/texture, glow/bloom, SSR, probes), code examples, and pipeline strategies. The goal is "fake" or "glow trace" lighting that *looks* close to ray-traced.

Floodlit race track at night. Instead of placing dozens of real lights for each lamp, you can make the track and surroundings glow via emissive materials and a bloom/glow post-process. This cheats light bouncing (the road appears lit) without true dynamic GI.

Step 1: Emissive Materials as "Lights"

Use **StandardMaterial** or **PBRMaterial** with a strong emissive component. An emissive material "radiates" its color or texture as if self-illuminated [1]. For example:

```
const mat = new BABYLON.PBRMaterial("mat", scene);
mat.emissiveColor = new BABYLON.Color3(1, 0.8, 0.2);    // bright warm glow
// or use an emissiveTexture:
// mat.emissiveTexture = new BABYLON.Texture("glow.png", scene);
const mesh = BABYLON.MeshBuilder.CreatePlane("lightPlane", { size:5 }, scene);
mesh.material = mat;
```

By default, PBR and Standard materials light up parts of a scene with *lights* in the scene. To make an object purely emissive (ignoring real lights), you can disable lighting on it (`standardMat.disableLighting = true`) or simply rely on its emissive color. A **lightmap** texture can also store pre-baked lighting: assign a texture to `pbrMaterial.lightmapTexture` to simulate indirect light [2]. In static scenes, you might bake GI into lightmaps or vertex colors ahead of time [2] [3].

Key APIs: `PBRMaterial.emissiveColor`, `PBRMaterial.emissiveTexture`, `StandardMaterial.emissiveColor/Texture`.
Example: make a glowing billboard:

```
mat.emissiveTexture = new BABYLON.Texture("sign_glow.png", scene);
mat.emissiveColor = new BABYLON.Color3(0.5, 0.5, 0.5);
mat.emissiveIntensity = 2;  // amplify glow (if using a PBRMaterial)
```

Step 2: Bloom/Glow Effects

Enhance emissive surfaces with bloom or glow so they *appear* bright. Babylon's **GlowLayer** captures emissive parts and blurs them to simulate bloom. For example:

```
var glow = new BABYLON.GlowLayer("glow", scene);
glow.intensity = 0.6;
// Now all meshes with emissiveColor/Texture will have a glow.
```

This makes bright objects bleed light onto surrounding pixels. A **lens flare system** or **post-process bloom** (using imageProcessingConfiguration) can further exaggerate bright spots. In practice, putting a glow on an emissive surface helps it look like a light source even though it's just a texture.

Step 3: Reflections (SSR & Probes)

Reflections are a big part of ray-traced look. Babylon supports **Screen-Space Reflections (SSR)** and **Reflection Probes** for this:

- **SSR Rendering Pipeline:** Babylon 5.51+ includes an SSR pipeline for real-time reflections [4]. You can enable it with:

  ```
  const ssr = new BABYLON.SSRRenderingPipeline("ssr", scene, {
      rayStepCount: 16,          // reflection ray steps
      reflectionSamples: 32,   // quality
      useJittering: true,
      enableSmoothReflections: true
  }, true);
  ```

  SSR will reflect visible surfaces and emissive objects in the screen view (standard/PBR materials only). It gives dynamic mirrors on shiny objects without ray tracing. Because SSR only sees what's on-screen, you may see black gaps; combining SSR with environment probes helps fill missing data.

- **ReflectionProbe:** Capture the scene (including emissives) into a cubemap. For example:

  ```
  const probe = new BABYLON.ReflectionProbe("probe", 128, scene);
  scene.meshes.forEach(m => probe.renderList.push(m));
  mesh.material.reflectionTexture = probe.cubeTexture;
  ```

  A probe placed in the scene captures nearby emissive meshes and geometry, producing static-ish reflections. This simulates "environment lighting" from emissive objects onto reflective materials.

- **Image-Based Lighting (IBL):** Use an HDR skybox or environment texture to light the scene. In PBR, `scene.environmentTexture` provides ambient light and reflections that blend with emissive surfaces.

Step 4: Simulating Global Illumination

True multi-bounce GI requires ray tracing or complex techniques. In Babylon we can approximate some GI with:

- **Baked Lightmaps/Light Probes:** Pre-calculate indirect light into textures or probe volumes. Babylon doesn't have built-in dynamic light probes, but you can bake indirect light from emissives into lightmaps or vertex colors. Use `material.lightmapTexture` for baked indirect lighting [2].

- **Reflective Shadow Maps (RSM GI):** Babylon's new experimental GI uses RSM (screen-space GI) [5]. This renders the scene from a light's POV (like a shadow map) but also stores flux for a bounce. It's not full GI (see Evgeni's note: *"not a full GI solution"* [5]), but it can brighten areas under emissives. To use it:

```
const rsm = new BABYLON.ReflectiveShadowMap(scene, light);
rsm.renderList.push(mesh);
const giManager = new BABYLON.GIRSMManager(scene, { width:512, height:
512 });
giManager.addGIRSM(new BABYLON.GIRSM(rsm));
giManager.enable = true;
```

(RSM requires additional setup and may not work with all post-processes [6] [5].)

- **Screen-Space AO/Curvature:** Use SSAO to darken creases and indirect corners. While ambient occlusion is the *opposite* of GI, it provides shading that makes scenes look more three-dimensional. Babylon's SSAO2 pipeline or curvature shaders help fake subtle light falloff in crevices [7].

- **Ambient Lighting:** Adjust `scene.ambientColor` or use a low-intensity directional light to lift dark areas subtly. Combined with emissives and bloom, this prevents fully black shadows.

Step 5: Soft Shadows and Scattering

Although our emissives are not casting real shadows, we can still fake soft shadowing and scattering:

- **Shadow Generator:** If you do add a few real lights (e.g. point lights at emissive objects), use `ShadowGenerator` with high map size and blur/dither to soften shadows. Soft shadows make the scene look more natural. For example:

```
const light = new BABYLON.PointLight("pl", mesh.position, scene);
const shadowGen = new BABYLON.ShadowGenerator(1024, light);
```

```
shadowGen.usePoissonSampling = true;
shadowGen.useBlurExponentialShadowMap = true;
```

- **Volumetric Light (God Rays):** For a "gloomy" or hazy look, `VolumetricLightScatteringPostProcess` can create shafts of light around emissive sources (god-rays).
- **Fog:** Light fog can simulate the effect of distant light scattering.

Combining these, a scene lit by emissive "lights" will still have gentle shading and depth cues. Use **SSAO** or **cavity effects** to darken crevices, and blur shadows for softness [7].

Step 6: Putting It All Together – A Custom Pipeline

Babylon 8+ allows fully custom render pipelines via the **Node Render Graph** [8]. You can build a post-processing chain like:

1. **Geometry Pass (G-Buffer):** Render normals, depth, and material IDs (enable `scene.enablePrePassRenderer()` or use GeometryBufferRenderer).
2. **Identify Emissives:** In a fullscreen shader, flood the buffer with light from emissive materials' color. For example, add the emissive color of bright pixels to a light accumulation buffer.
3. **Reflection Pass:** Apply SSR or reflection probes. SSR uses the G-Buffer to trace rays in screen-space; add that to the buffer.
4. **Indirect Bounce (Optional):** If using RSM GI, inject its contribution.
5. **Combine Lighting:** Draw final colors by combining diffuse, emissive, specular (some data comes from G-Buffer or PBR shader).
6. **Post-Processing:** Add bloom/glow, tone mapping, and possibly FXAA or dithering.

In Babylon 6/7, you can approximate this with the built-in pipelines: `DefaultRenderingPipeline` or Node Material blocks for custom shaders. The key is to blur and bleed light from bright emissive areas onto surrounding pixels (for example, a Gaussian blur of the emissive mask adds soft light to nearby surfaces). Using the **GlowLayer** and **ImageProcessingConfiguration.bloom** steps outside the main render can simulate part of this effect.

Limitations vs Real Raytracing

This "emissive+SSR" approach has tradeoffs. Real ray tracing computes *actual* light paths, whereas our method **only simulates** them. Limitations include:

- **No True Bounce:** Unless you bake or use RSM, emissive objects won't dynamically light occluded areas. (As one forum answer puts it: casting light from an emissive object **"would mostly require ray tracing to work well"** [9].)
- **Screen-Space Gaps:** SSR only reflects what's on screen; off-screen lights/reflections won't appear.
- **Incomplete GI:** RSM GI is not full GI (Babylon's docs warn it's limited [5]). Scenes with complex indirect lighting (color bleeding, caustics) won't be fully accurate.
- **Performance:** Adding many fake lights or large probes can hit performance; the Node Graph is powerful but still GPU-bound.
- **Precision:** Without ray casting, thin or small bright objects may not light surroundings correctly.

That said, for many scenes emissive tricks are “good enough” and far faster than real ray tracing on current WebGL hardware. Users on Babylon forums have built static scenes using only emissive textures, baked lightmaps, and HDR skyboxes to achieve a lit look [3]. Others have recommended using *deferred lighting* or SSAO to hide the limitations [10] [7].

Clever Names for “Fake Raytracer”

For fun, you can name your tech something ironic like:
- **FakeRays** or **FallRay** (fall-back of ray)
- **GlowTrace** or **EmissiTrace**
- **RadiosityLite** or **CheapoTracer**
- **BloomTracer** or **ScreenTrace**

These suggest imitation of ray tracing via bloom/glow and screen-space tricks. Babylon’s developers once joked about “GlowLight” and the later addition of **Area Lights** in version 8 (truly emissive rectangles) brings this concept closer to real lighting [11].

References

We’ve leveraged Babylon.js features (emissive materials [1], glow layer, SSR [4], probes, lightmaps [2]) and community insights (casting light from emissives [9], GI limitations [5]) to craft this approach. Many examples on the forum (e.g. a night scene lit by emissives [12]) illustrate these techniques. While not true ray tracing, this pipeline can produce rich, dynamic visuals with glow and screen-space lighting effects that often *pass* for GI at interactive frame rates.

---

[1] Babylon.js Emissive Properties
https://www.tutorialspoint.com/babylonjs/babylonjs_emissive.htm

[2] BabylonJS Documentation
http://grideasy.github.io/overviews/Physically_Based_Rendering_Master

[3] Static Lighting and Reflection - Questions - Babylon.js
https://forum.babylonjs.com/t/static-lighting-and-reflection/19029

[4] Screen Space Reflection (SSR) overhaul - Demos and projects - Babylon.js
https://forum.babylonjs.com/t/screen-space-reflection-ssr-overhaul/39089

[5] How to use RSM Global Illumination? - Questions - Babylon.js
https://forum.babylonjs.com/t/how-to-use-rsm-global-illumination/47702

[6] ReflectiveShadowMap GI disappears when using SSAO, SSR or MotionBlur - Bugs - Babylon.js
https://forum.babylonjs.com/t/reflectiveshadowmap-gi-disappears-when-using-ssao-ssr-or-motionblur/49538

[7] Fake lighting on unlit materials - Questions - Babylon.js
https://forum.babylonjs.com/t/fake-lighting-on-unlit-materials/55876

[8] [11] Introducing Babylon.js 8.0. Our mission is to build one of the most... | by Babylon.js | Medium
https://babylonjs.medium.com/introducing-babylon-js-8-0-77644b31e2f9

9 Casting light from Emissive Object - Questions - Babylon.js
https://forum.babylonjs.com/t/casting-light-from-emissive-object/25487

10 12 Floodlit night effect - Questions - Babylon.js
https://forum.babylonjs.com/t/floodlit-night-effect/57470