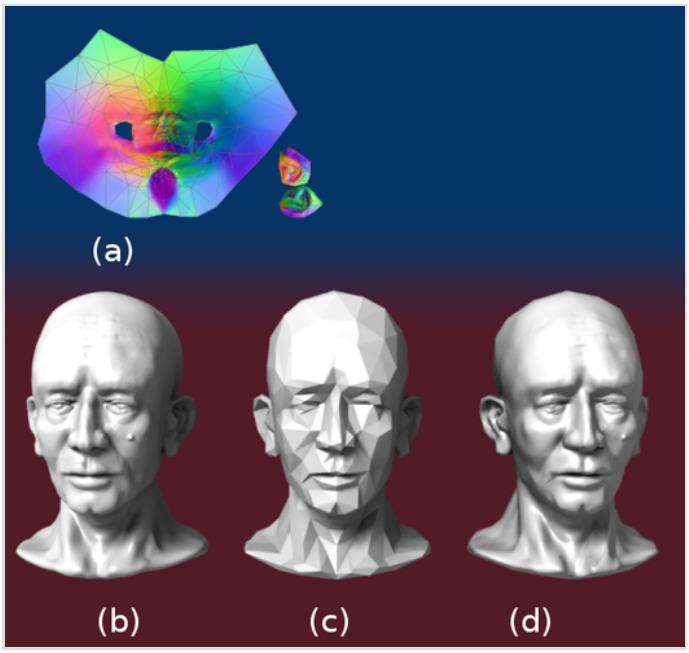
WIKIPEDIA

Normal mapping

In 3D computer graphics, **normal mapping**, or **Dot3 bump mapping**, is a texture mapping technique used for faking the lighting of bumps and dents – an implementation of bump mapping. It is used to add details without using more polygons.[1] A common use of this technique is to greatly enhance the appearance and details of a low polygon model by generating a normal map from a high polygon model or height map.

Normal maps are commonly stored as regular RGB images where the RGB components correspond to the X, Y, and Z coordinates, respectively, of the surface normal.



Normal mapping used to re-detail simplified meshes. Normal map (a) is baked from 78,642 triangle model (b) onto 768 triangle model (c). This results in a render of the 768 triangle model, (d).

History

In 1978 Jim Blinn described how the normals of a surface could be perturbed to make geometrically flat faces have a detailed appearance.[2] The idea of taking geometric details from a high polygon model was introduced in "Fitting Smooth Surfaces to Dense Polygon Meshes" by Krishnamurthy and Levoy, Proc. SIGGRAPH 1996,[3] where this approach was used for creating displacement maps over nurbs. In 1998, two papers were presented with key ideas for transferring details with normal maps from high to low polygon meshes: "Appearance Preserving Simplification", by Cohen et al. SIGGRAPH 1998,[4] and "A general method for preserving attribute values on simplified meshes" by Cignoni et al. IEEE Visualization '98.[5] The former introduced the idea of storing surface normals directly in a texture, rather than displacements, though it required the low-detail model to be generated by a particular constrained simplification algorithm. The latter presented a simpler approach that decouples the high and low polygonal mesh and allows the recreation of any attributes of the high-detail model (color, texture coordinates, displacements, etc.) in a way that is not dependent on how the low-detail model was created. The combination of storing normals in a texture, with the more general creation process is still used by most currently available tools.

Spaces

The orientation of coordinate axes differs depending on the space in which the normal map was encoded. A straightforward implementation encodes normals in object space so that the red, green, and blue components correspond directly with the X, Y, and Z coordinates. In object space, the coordinate system is constant.

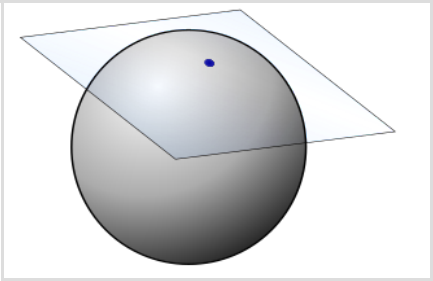
However, object-space normal maps cannot be easily reused on multiple models, as the orientation of the surfaces differs. Since color texture maps can be reused freely, and normal maps tend to correspond with a particular texture map, it is desirable for artists that normal maps have the same property.

Normal map reuse is made possible by encoding maps in tangent space. The tangent space is a vector space, which is tangent to the model's surface. The coordinate system varies smoothly (based on the derivatives of position with respect to texture coordinates) across the surface.

A texture map (left). The corresponding normal map in tangent space (center). The normal map applied to a sphere in object space (right).

Tangent space normal maps can be identified by their dominant purple color, corresponding to a vector facing directly out from the surface. See Calculation.

Calculating tangent spaces



A pictorial representation of the tangent space of a single point x on a sphere

Surface normals are used in computer graphics primarily for the purposes of lighting, through mimicking a phenomenon called specular reflection. Since the visible image of an object is the light bouncing off of its surface, the light information obtained from each point of the surface can instead be computed on its tangent space at that point.

For each tangent space of a surface in 3-dimensional space, there are two vectors which are perpendicular to every vector of the tangent space. These vectors are called normal vectors, and choosing between these two vectors provides a description on how the surface is oriented at that point, as the light information depends on the angle of incidence between the ray r and the normal vector n, and the light will only be visible if $\langle r, n \rangle > 0$. In such a case, the reflection s of the ray with direction r along the normal vector n is given by

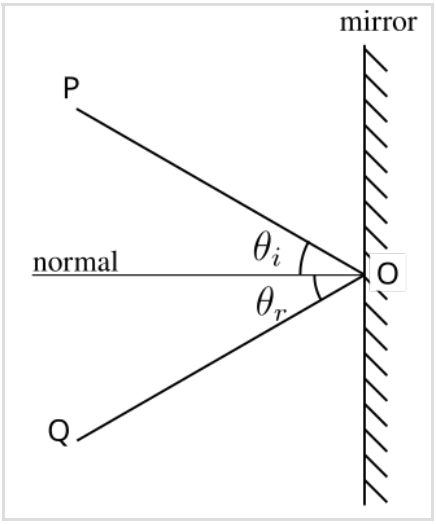
$$s = r - 2\langle n, r \rangle n$$

Intuitively, this just means that you can only see the outward face of an object if you're looking from the outside, and only see the inward face if you're looking from the inside. Note that the light information is local, and so the surface does not necessarily need to be orientable as a whole. This is why even though spaces such as the Möbius strip and the Klein bottle are non-orientable, it is still possible to visualize them.

Normals can be specified with a variety of coordinate systems. In computer graphics, it is useful to compute normals relative to the tangent plane of the surface. This is useful because surfaces in

applications undergo a variety of transforms, such as in the process of being rendered, or in skeletal animations, and so it is important for the normal vector information to be preserved under these transformations. Examples of such transforms include transformation, rotation, shearing and scaling, perspective projection,[6] or the skeletal animations on a finely detailed character.

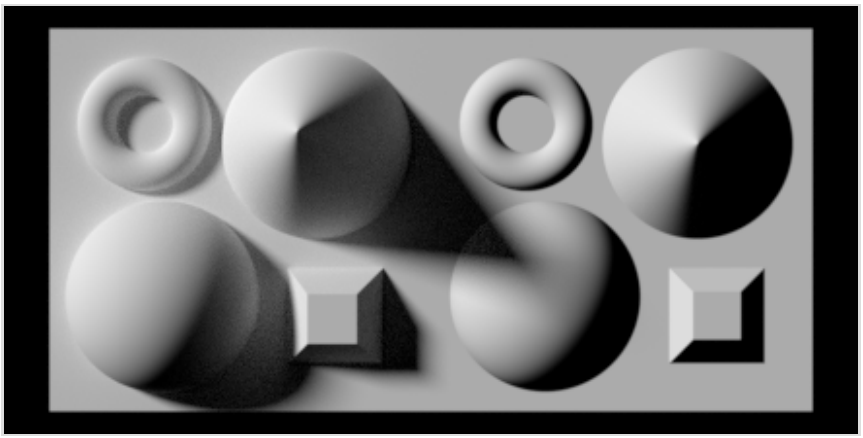
For the purposes of computer graphics, the most common representation of a surface is a triangulation, and as a result, the tangent plane at a point can be obtained through interpolating between the planes that contain the triangles that each intersect that point. Similarly, for parametric surfaces with tangent spaces, the parametrizations will yield partial derivatives, and these derivatives can be used as a basis of the tangent spaces at every point.



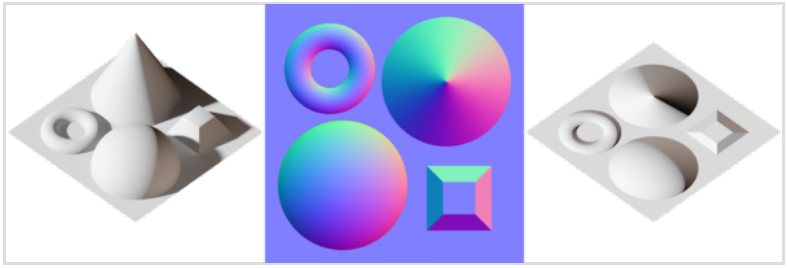
A graphic depicting how the normal vector determines the reflection of a ray

In order to find the perturbation in the normal the tangent space must be correctly calculated.[7] Most often the normal is perturbed in a fragment shader after applying the model and view matrices. Typically the geometry provides a normal and tangent. The tangent is part of the tangent plane and can be transformed simply with the linear part of the matrix (the upper 3x3). However, the normal needs to be transformed by the inverse transpose. Most applications will want bitangent to match the transformed geometry (and associated UVs). So instead of enforcing the bitangent to be perpendicular to the tangent, it is generally preferable to transform the bitangent just like the tangent. Let t be tangent, b be bitangent, n be normal, M_{3x3} be the linear part of model matrix, and V_{3x3} be the linear part of the view matrix.

$$
\begin{aligned}
t' &= t \times M_{3x3} \times V_{3x3} \\
b' &= b \times M_{3x3} \times V_{3x3} \\
n' &= n \times (M_{3x3} \times V_{3x3})^{-1T} = n \times M_{3x3}^{-1T} \times V_{3x3}^{-1T}
\end{aligned}
$$


Rendering using the normal mapping technique. On the left, several solid meshes. On the right, a plane surface with the normal map computed from the meshes on the left.

Calculation



Example of a normal map (center) with the scene it was calculated from (left) and the result when applied to a flat surface (right). This map is encoded in tangent space.

To calculate the Lambertian (diffuse) lighting of a surface, the unit vector from the shading point to the light source is dotted with the unit vector normal to that surface, and the result is the intensity of the light on that surface. Imagine a polygonal model of a sphere - you can only approximate the shape of the surface. By using a 3-channel bitmap textured across the model, more detailed normal vector information can be encoded. Each channel in the bitmap corresponds to a spatial dimension (X, Y and Z). These spatial dimensions are relative to a constant coordinate system for object-space normal maps, or to a smoothly varying coordinate system (based on the derivatives of position with respect to texture coordinates) in the case of tangent-space normal maps. This adds much more detail to the surface of a model, especially in conjunction with advanced lighting techniques.

Unit Normal vectors corresponding to the u,v texture coordinate are mapped onto normal maps. Only vectors pointing towards the viewer (z: 0 to -1 for Left Handed Orientation) are present, since the vectors on geometries pointing away from the viewer are never shown. The mapping is as follows:

```
X: −1 to +1 :  Red:     0 to 255
Y: −1 to +1 :  Green:   0 to 255
Z:  0 to −1 :  Blue:  128 to 255
```

```
                light green    light yellow
dark cyan       light blue     light red
dark blue       dark magenta
```

- A normal pointing directly towards the viewer (0,0,-1) is mapped to (128,128,255). Hence the parts of object directly facing the viewer are light blue. The most common color in a normal map.
- A normal pointing to top right corner of the texture (1,1,0) is mapped to (255,255,128). Hence the top-right corner of an object is usually light yellow. The brightest part of a color map.
- A normal pointing to right of the texture (1,0,0) is mapped to (255,128,128). Hence the right edge of an object is usually light red.
- A normal pointing to top of the texture (0,1,0) is mapped to (128,255,128). Hence the top edge of an object is usually light green.
- A normal pointing to left of the texture (-1,0,0) is mapped to (0,128,128). Hence the left edge of an object is usually dark cyan.
- A normal pointing to bottom of the texture (0,-1,0) is mapped to (128,0,128). Hence the bottom edge of an object is usually dark magenta.
- A normal pointing to bottom left corner of the texture (-1,-1,0) is mapped to (0,0,128). Hence the bottom-left corner of an object is usually dark blue. The darkest part of a color map.

Since a normal will be used in the dot product calculation for the diffuse lighting computation, we can see that the {0, 0, −1} would be remapped to the {128, 128, 255} values, giving that kind of sky blue

color seen in normal maps (blue (z) coordinate is perspective (deepness) coordinate and RG-xy flat coordinates on screen). {0.3, 0.4, −0.866} would be remapped to the ({0.3, 0.4, −0.866}/2+{0.5, 0.5, 0.5})*255={0.15+0.5, 0.2+0.5, -0.433+0.5}*255={0.65, 0.7, 0.067}*255={166, 179, 17} values ($0.3^2 + 0.4^2 + (-0.866)^2 = 1$). The sign of the z-coordinate (blue channel) must be flipped to match the normal map's normal vector with that of the eye (the viewpoint or camera) or the light vector. Since negative z values mean that the vertex is in front of the camera (rather than behind the camera) this convention guarantees that the surface shines with maximum strength precisely when the light vector and normal vector are coincident.[8]

Normal mapping in video games

Interactive normal map rendering was originally only possible on PixelFlow, a parallel rendering machine built at the University of North Carolina at Chapel Hill. It was later possible to perform normal mapping on high-end SGI workstations using multi-pass rendering and framebuffer operations[9] or on low end PC hardware with some tricks using paletted textures. However, with the advent of shaders in personal computers and game consoles, normal mapping became widespread in the early 2000s, with some of the first games to implement it being Evolva (2000), Giants: Citizen Kabuto, and Virtua Fighter 4 (2001).[10][11] Normal mapping's popularity for real-time rendering is due to its good quality to processing requirements ratio versus other methods of producing similar effects. Much of this efficiency is made possible by distance-indexed detail scaling, a technique which selectively decreases the detail of the normal map of a given texture (cf. mipmapping), meaning that more distant surfaces require less complex lighting simulation. Many authoring pipelines use high resolution models baked into low/medium resolution in-game models augmented with normal maps.

Basic normal mapping can be implemented in any hardware that supports palettized textures. The first game console to have specialized normal mapping hardware was the Sega Dreamcast. However, Microsoft's Xbox was the first console to widely use the effect in retail games. Out of the sixth generation consoles, only the PlayStation 2's GPU lacks built-in normal mapping support, though it can be simulated using the PlayStation 2 hardware's vector units. Games for the Xbox 360 and the PlayStation 3 rely heavily on normal mapping and were the first game console generation to make use of parallax mapping. The Nintendo 3DS has been shown to support normal mapping, as demonstrated by *Resident Evil: Revelations* and *Metal Gear Solid 3: Snake Eater*.

See also

- Reflection (physics)
- Ambient occlusion
- Depth map
- Baking (computer graphics)
- Tessellation (computer graphics)
- Bump mapping
- Displacement mapping

References

1. "LearnOpenGL - Normal Mapping" (https://learnopengl.com/Advanced-Lighting/Normal-Mapping). *learnopengl.com*. Retrieved 2024-05-21.
2. Blinn. *Simulation of Wrinkled Surfaces (https://www.microsoft.com/en-us/research/wp-content/uploads/1978/01/p286-blinn.pdf)*, Siggraph 1978
3. Krishnamurthy and Levoy, *Fitting Smooth Surfaces to Dense Polygon Meshes (http://www.graphics.stanford.edu/papers/surfacefitting/)*, SIGGRAPH 1996
4. Cohen et al., Appearance-Preserving Simplification (http://www.cs.unc.edu/~geom/APS/APS.pdf), SIGGRAPH 1998 **(PDF)**
5. Cignoni et al., A general method for preserving attribute values on simplified meshes (http://vcg.isti.cnr.it/publications/papers/rocchini.pdf), IEEE Visualization 1998 **(PDF)**
6. Akenine-Möller, Tomas; Haines, Eric; Hoffman, Naty; Pesce, Angelo; Iwanicki, Michał; Hillaire, Sébastien (2018). *Real-Time Rendering 4th Edition* (https://www.realtimerendering.com/) (4 ed.). Boca Raton, FL, USA: A K Peters/CRC Press. p. 57. ISBN 978-1-13862-700-0. Retrieved 2 August 2024.
7. Mikkelsen, Simulation of Wrinkled Surfaces Revisited (http://image.diku.dk/projects/media/morten.mikkelsen.08.pdf), 2008 **(PDF)**
8. "LearnOpenGL - Normal Mapping" (https://learnopengl.com/Advanced-Lighting/Normal-Mapping). *learnopengl.com*. Retrieved 2021-10-19.
9. Heidrich and Seidel, Realistic, Hardware-accelerated Shading and Lighting (http://www.cs.ubc.ca/~heidrich/Papers/Siggraph.99.pdf) Archived (https://web.archive.org/web/20050129211042/http://www.cs.ubc.ca/~heidrich/Papers/Siggraph.99.pdf) 2005-01-29 at the Wayback Machine, SIGGRAPH 1999 **(PDF)**
10. "Virtua Fighter 4" (https://segaretro.org/Virtua_Fighter_4). *Sega Retro*. 2023-11-30. Retrieved 2024-03-03.
11. "Tecnologías gráficas en los juegos" (https://as.com/meristation/2006/08/01/reportajes/1154449800_036749.html). *Meristation* (in Spanish). 2012-04-18. Retrieved 2024-03-03.

External links

- Normal Map Tutorial (https://web.archive.org/web/20160820195558/http://www.falloutsoftware.com/tutorials/gl/normal-map.html) Per-pixel logic behind Dot3 Normal Mapping
- NormalMap-Online (https://cpetry.github.io/NormalMap-Online) Free Generator inside Browser
- Normal Mapping on sunandblackcat.com (https://web.archive.org/web/20150503205556/http://sunandblackcat.com/tipFullView.php?l=eng&topicid=7)
- Blender Normal Mapping (https://web.archive.org/web/20160827174956/https://www.blender.org/manual/render/blender_render/textures/influence/material/bump_and_normal.html)
- Normal Mapping with paletted textures (https://web.archive.org/web/20050308073824/http://vcg.isti.cnr.it/activities/geometryegraphics/bumpmapping.html) using old OpenGL extensions.
- Normal Map Photography (http://zarria.net/nrmphoto/nrmphoto.html) Creating normal maps manually by layering digital photographs
- Normal Mapping Explained (http://www.3dkingdoms.com/tutorial.htm)
- Simple Normal Mapper (https://sourceforge.net/projects/simplenormalmapper/) Open Source normal map generator