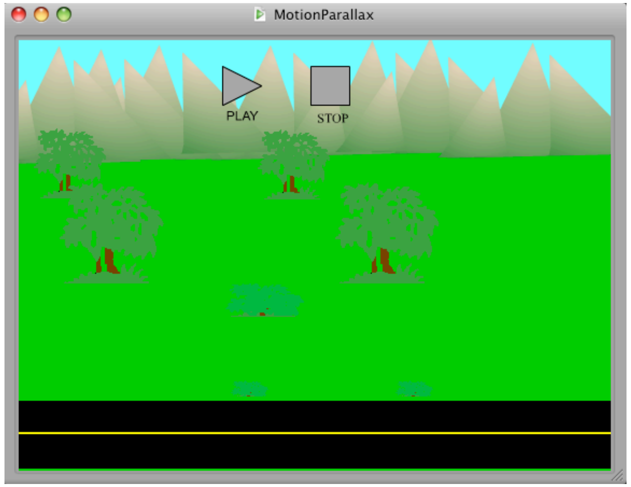
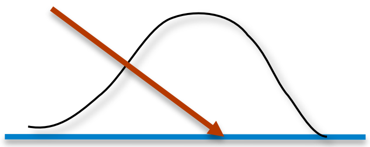
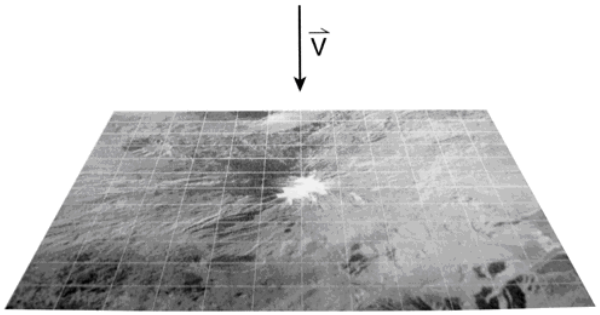
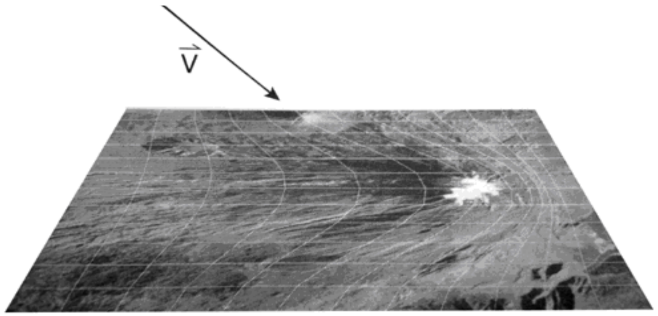
Parallax Mapping

- Given: coarse geometry + 2D texture + detailed height map
- Problem with bump / normal mapping:
  - Only the lighting is affected – the image of the texture remains unchanged, regardless of the viewing direction
  - Motion parallax: near / distant objects shift very differently relative to one another
- Extreme example:

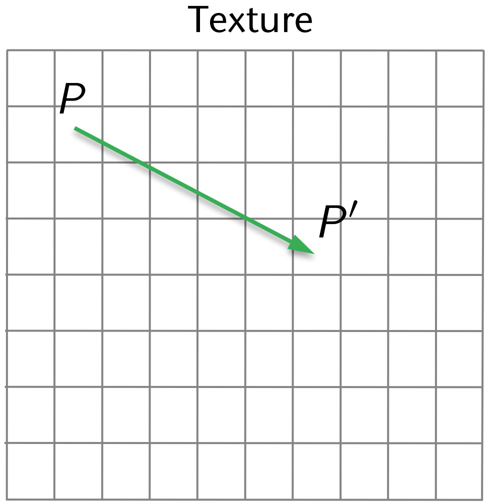


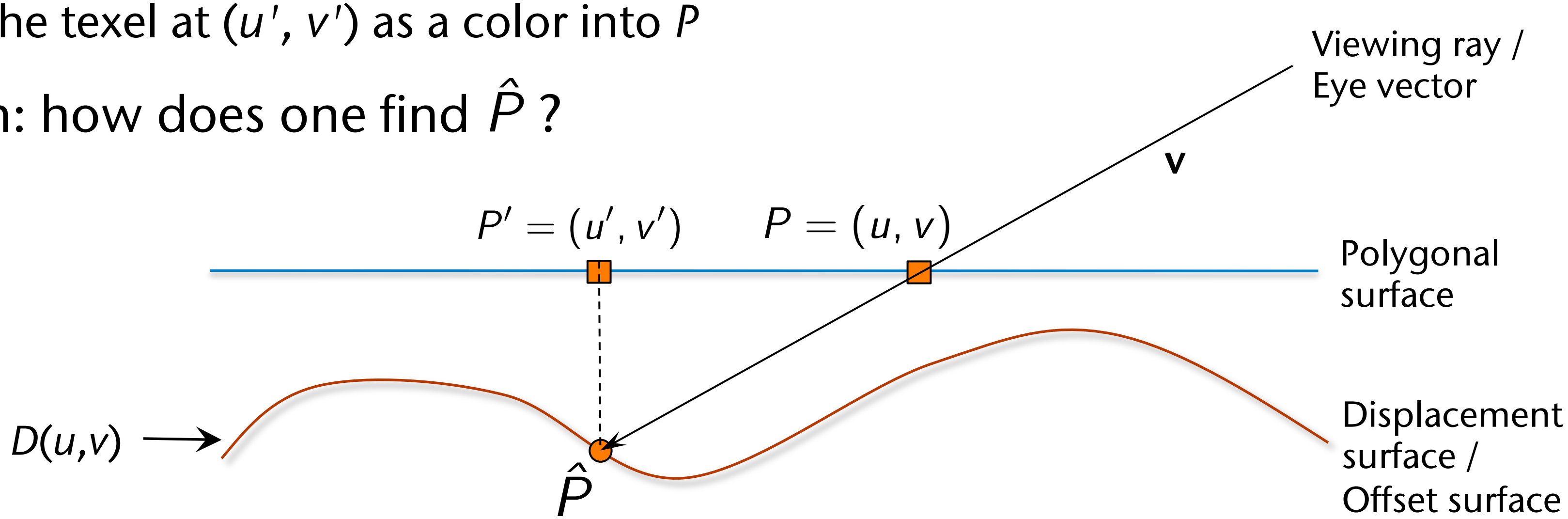




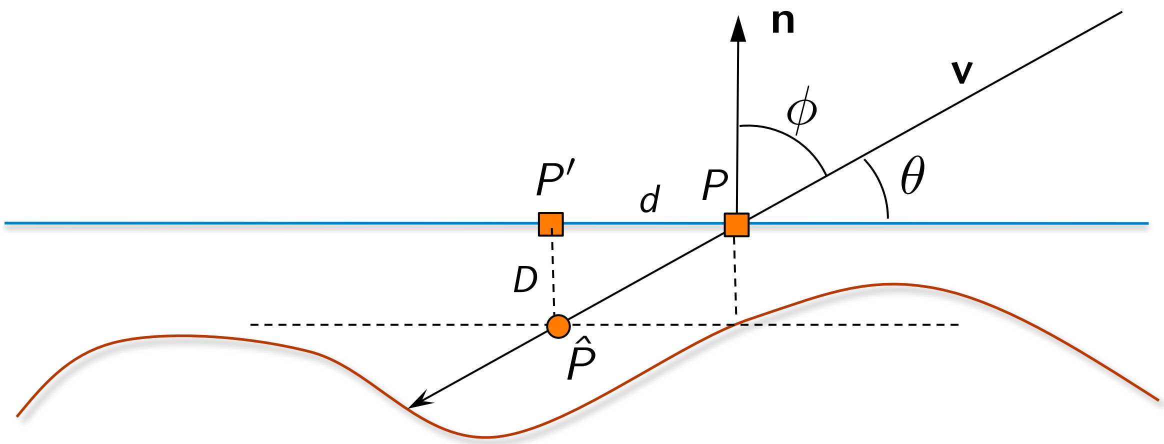


- Goal: "fake" motion parallax of *detailed* offset surface, although we only render *coarse* polygonal geometry
- The general task in parallax mapping:
  - Assume that scan line conversion is at pixel P
  - Determine point $\hat{P}$ that *would* be seen along **v**
  - Project $\hat{P}$ onto polygonal surface $\rightarrow P'$
  - Write the texel at (u', v') as a color into P
- Problem: how does one find $\hat{P}$?



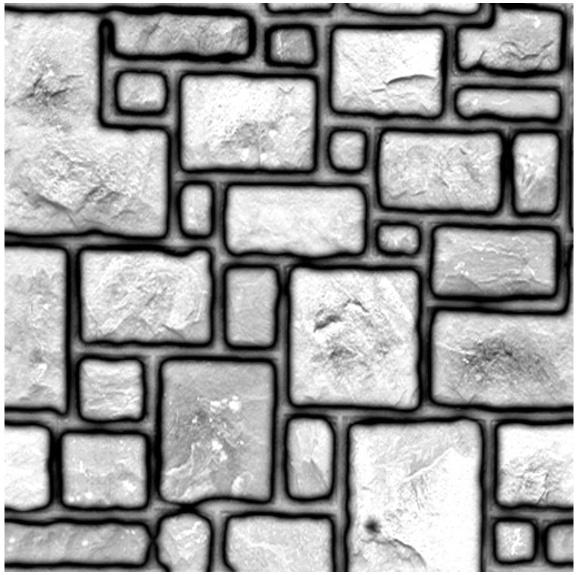


- Simplest idea:
  - We know the height $D = D(u,v)$ at point $P = P(u,v)$
  - Use this as an approximation of $D(u',v')$ in point $P' = P'(u,v)$
  - $\frac{D}{d} = \tan\theta = \frac{\sin\theta}{\cos\theta} = \frac{\cos\phi}{\sin\phi} = \frac{|\mathbf{nv}|}{|\mathbf{n}\times\mathbf{v}|}$



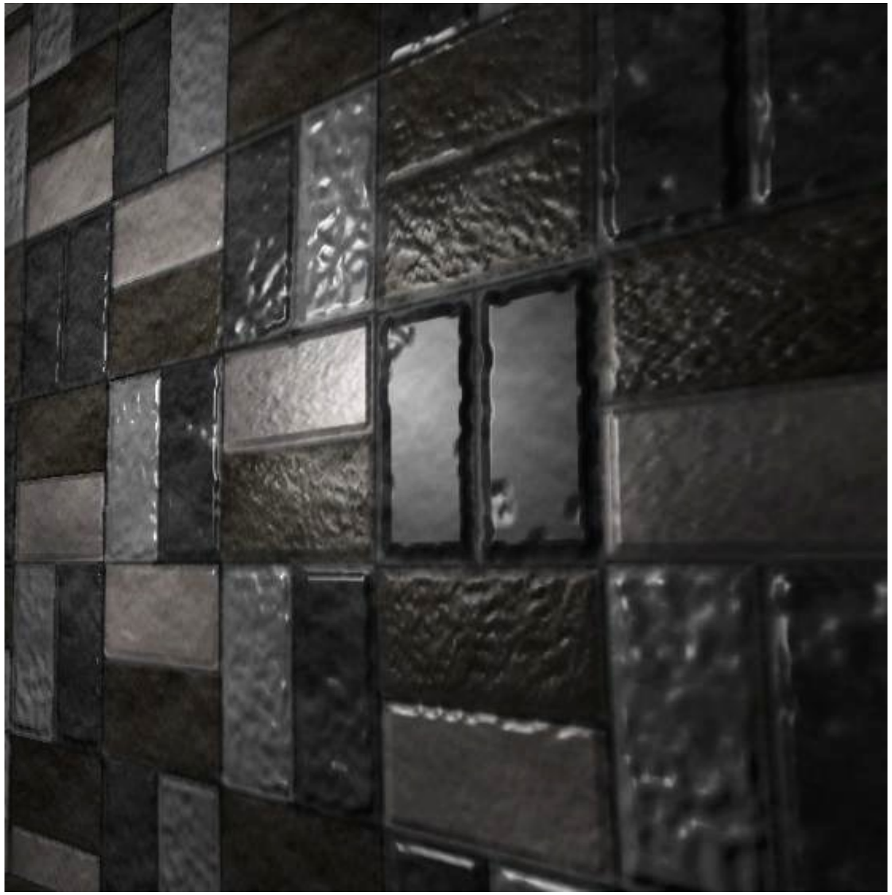
- Storage:
  - Put the image in the RGB channels of the texture
  - Put the heightmap in the alpha channel
- Process at rendering time:
  - Compute P' (see previous slide)
  - Calculate (u', v') of $P' \longrightarrow$ lookup texel
  - Perturb normal by bump mapping (see CG1)
    - Note: today one can calculate directional derivatives for D_u and D_v "on the fly" (needed in bump mapping algo)
  - Evaluate Phong model with texel color and perturbed normal

RGB

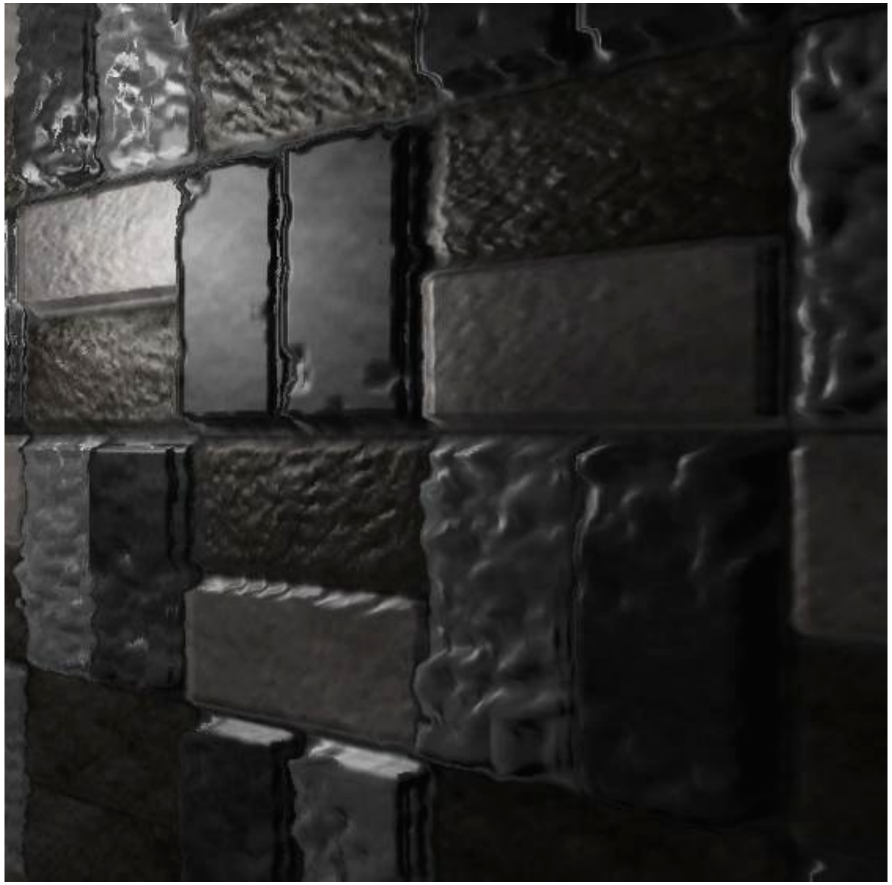


Alpha

Example



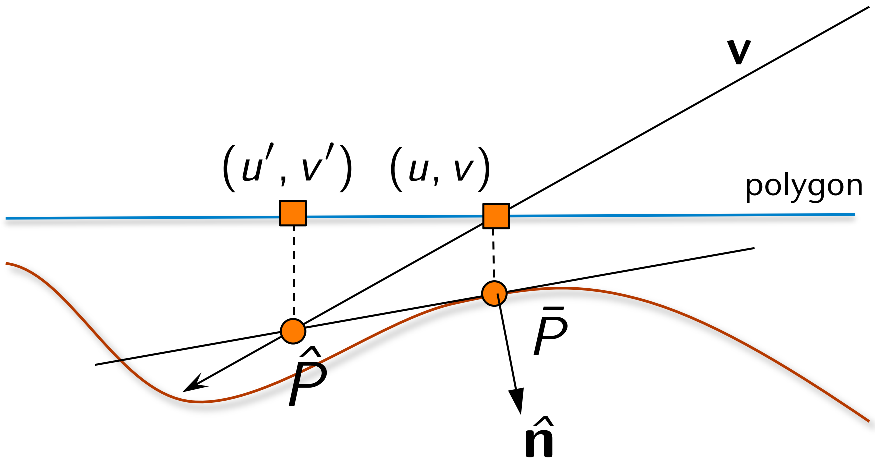
Normal Bump Mapping



Parallax Mapping
(For demonstration purposes,
parallax is strongly exaggerated here)

- Improvement: [Premecz, 2006]
  - Let $\bar{P} = (u, v, D)$ with $D = D(u,v)$
  - Approximate the heightmap in $\bar{P}$ through a plane (similar to bump mapping)
  - Calculate the point of intersection between that plane and the view vector:

$$\hat{\mathbf{n}} \left(\begin{pmatrix} u \\ v \\ 0 \end{pmatrix} + t\mathbf{v} - \begin{pmatrix} u \\ v \\ D \end{pmatrix} \right) = 0$$



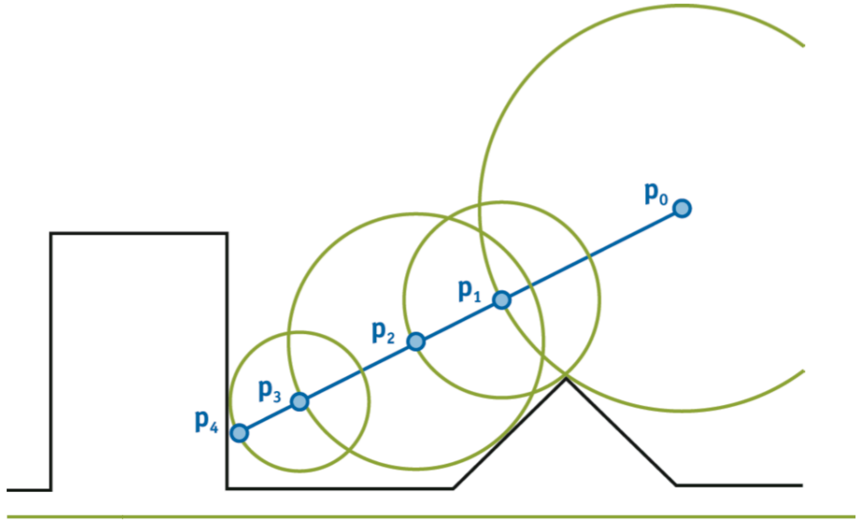
  - Solve for t
  - $\begin{pmatrix} u' \\ v' \end{pmatrix} = \begin{pmatrix} u \\ v \end{pmatrix} + t\mathbf{v}'$, with $\mathbf{v}' = \mathbf{v}$ projected into polygon's plane
- Additional (closely related) ideas: iteration, higher approximation of the heightmap

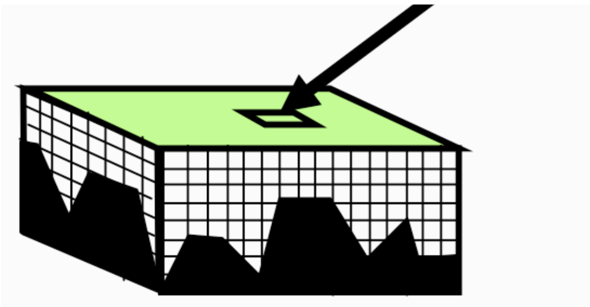
Thesis ...

Alternatives

[Donnelly, 2005]

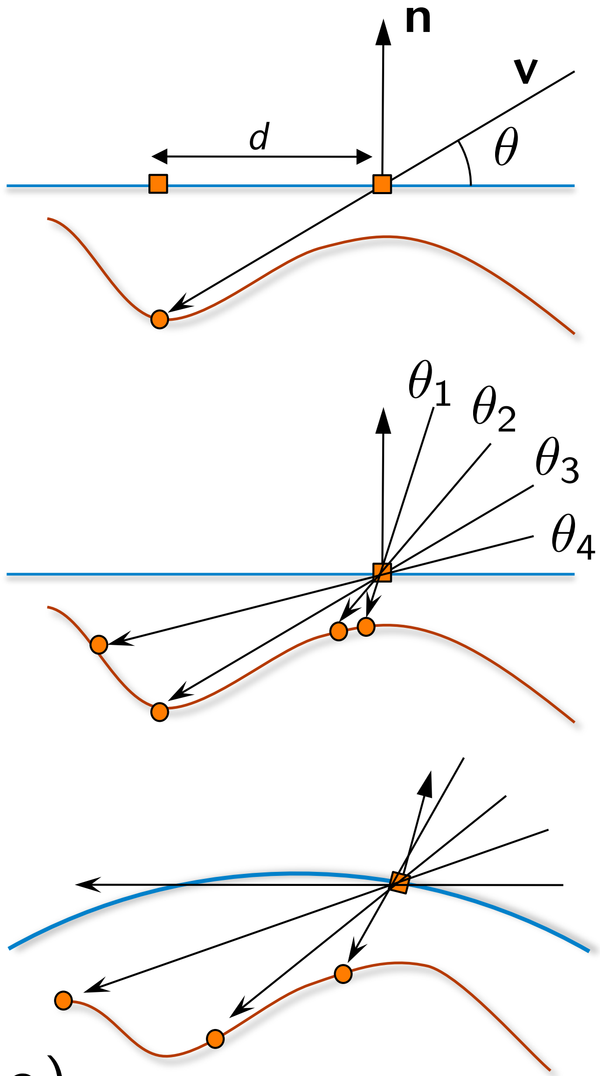
- Do sphere tracing along the view vectors, until you hit the offset surface
  - If the heightmap contains heights that are not too large, it is sufficient to begin relatively close underneath/ above the plane of reference
  - If the angle of the view vector is not too acute, then a few steps are sufficient
- For a layer underneath the plane of reference, save the smallest distance to the offset surface for every cell





View-Dependent Displacement Mapping (VDM)

- Idea: precompute all possible texture coordinate displacements for all possible situations
- In practice:
  - Parameterize the viewing vector by (θ, ϕ) in the local coordinate system of the polygon
  - Precompute the texture displacement for all (u,v) and all possible (θ, ϕ)
    - Ray casting of an explicit, temporarily generated mesh
  - Carry out the whole for a set of *possible* curvatures c of the base surface
- Results in a 5-dim. "texture" (LUT): $d(u, v, \theta, \phi, c)$



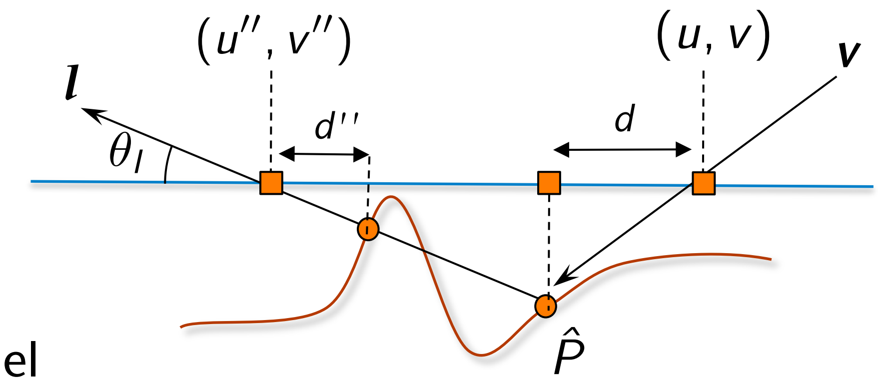
- Advantage: results in a correct silhouette
  - Reason: $d(\, u, v, \theta, \phi, c \,) = -1$ for many parameters near the silhouette
  - These are the pixels that lie outside of the silhouette!
- Further enhancement: self shadowing
  - Idea is similar to ray tracing: use "shadow rays"

1. Determine $\hat{P}$ from D and θ, ϕ (just like before) → (u,v) displacement d
2. Determine vektor $\boldsymbol{l}$ from $\hat{P}$ to the light source; and calc θ_l, ϕ_l from that
3. Determine $P'' = (u'', v'')$ from $\hat{P}$ and θ_l and ϕ_l
4. Make lookup in our "texture" $D \rightarrow d''$
5. Test:

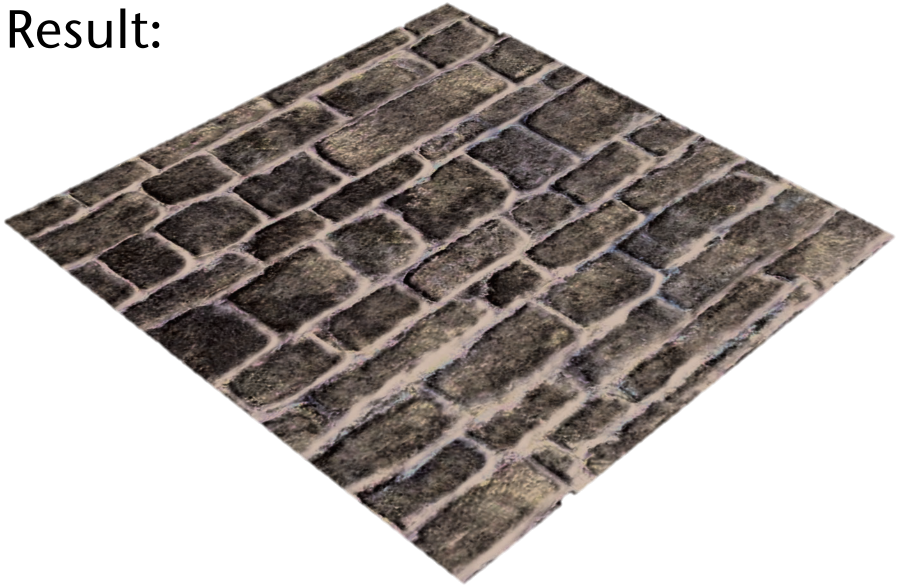
$$d'' + d < \|(u'', v'') - (u, v)\|$$

→ pixel (u,v) is in shadow

→ don't add light source $\boldsymbol{l}$ in Phong model



- Result:

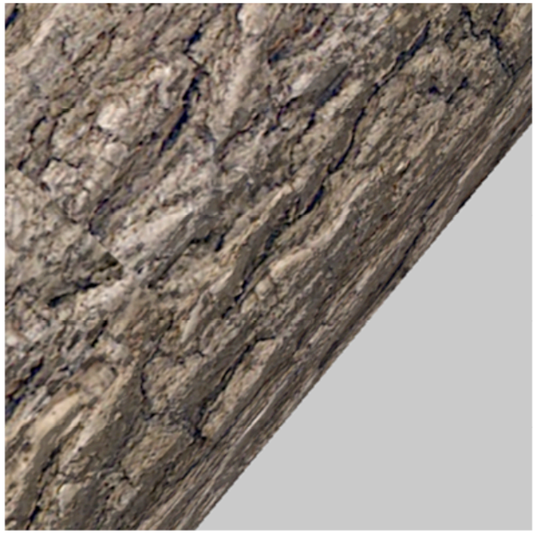


Bump Mapping

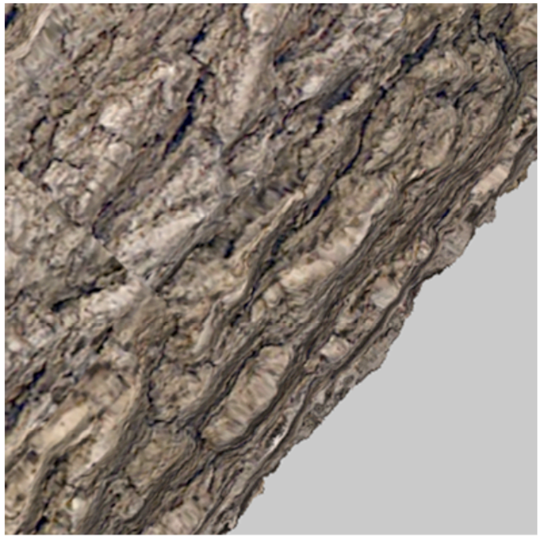
VDM

- Names:
  - Steep parallax mapping, parallax occlusion mapping, horizon mapping, view-dependent displacement mapping, ...
  - There are still many other variants ...
  - "Name ist Schall und Rauch!" ("A name is but noise and smoke!")

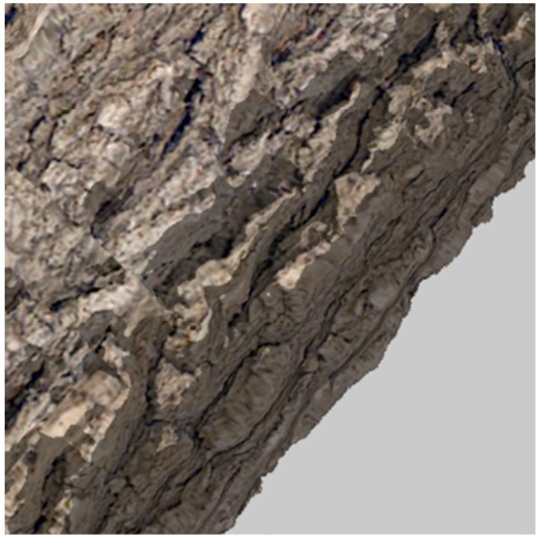
More Results



Bump mapping



Standard VDM



VDM with self-shadowing

All Examples Were Rendered with VDM