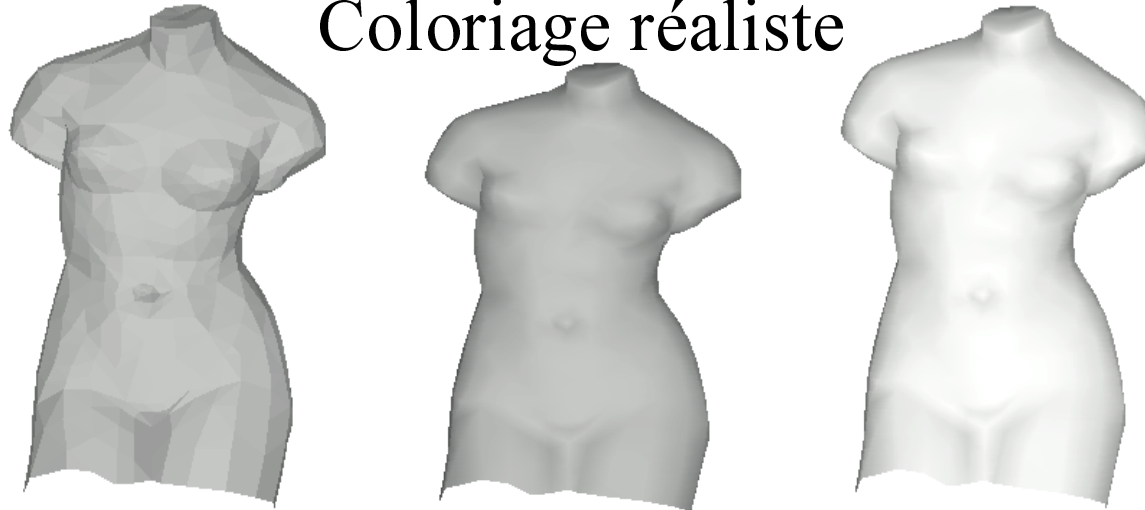


Valeur C et DEA
Conception d'applications multimédia

Synthèse d'images

Coloriage réaliste



Alexandre Topol

Département d'informatique
Conservatoire National des Arts et Métiers

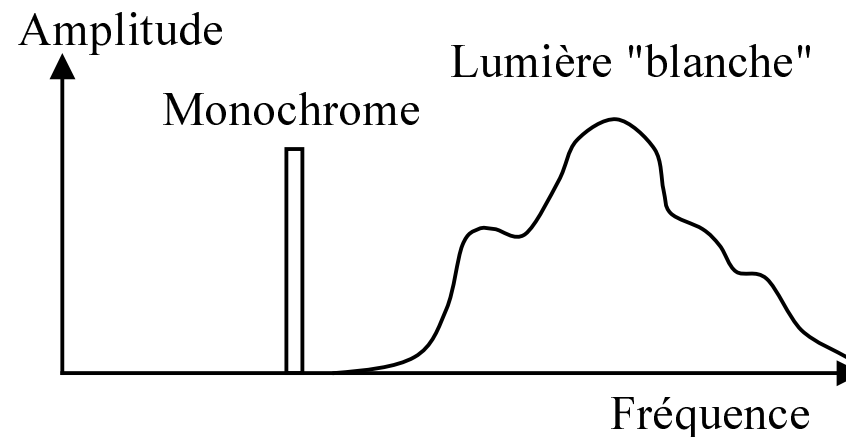
2001-2002

8. Coloriage réaliste

- 8.1. Un peu de physique ...
- 8.2. Le coloriage à plat
- 8.3. Interpolation de Gouraud
- 8.4. Modèle de Phong Bui-Tuong
- 8.5. Modèles de micro-facettes
- 8.6. Textures

8.1. Un peu de physique ...

- Couleur des corps diffusifs : 3 paramètres
 - Le spectre de la lumière qu'il reçoit
 - Les propriétés d'absorption du corps
 - Les propriétés de perception de l'œil
- Source lumineuse
 - Spectre d'émission



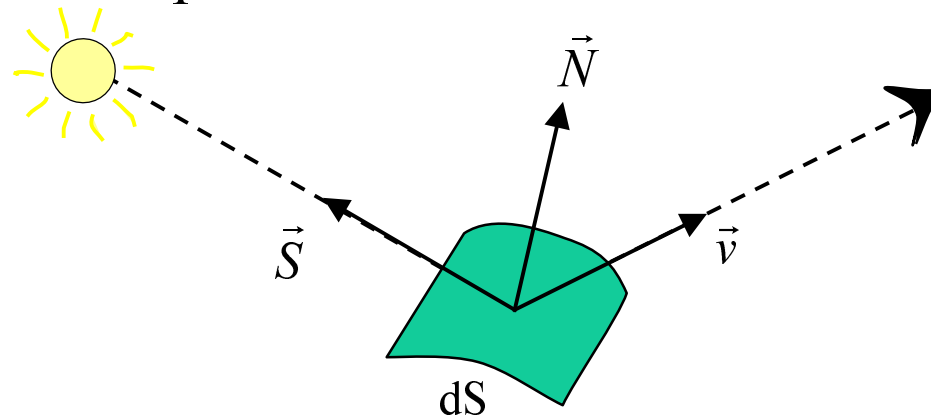
- Source ponctuelle ou non => cohérence spatiale

8.1. Un peu de physique ...

- Interaction lumière / objet
 - une partie de la puissance lumineuse reçue est absorbée par l'objet et convertie en chaleur
 - une partie est réfléchie par la surface
 - le reste est transmis à l'intérieur de l'objet (réfraction)
- Lumière ambiante : produit des multiples réflexions dans la scène

8.1. Un peu de physique ...

- Deux types de réflexion :
 - Diffuse : La surface de l'objet "réagit". La lumière est ré-émise dans toutes les directions. Sa couleur est affectée.
 - Spéculaire : La surface de l'objet ne "réagit" pas. La lumière est ré-émise selon l'angle d'incidence. Sa couleur n'est pas affectée.



8.1. Un peu de physique ...

- Les 2 composantes sont toujours plus ou moins présentes (exemple : papier \neq aluminium)
- Loi de Lambert pour la réflexion diffuse
 - I_{diff} = Intensité réfléchie
 - I_p = Intensité de la source ponctuelle
 - K_d = coefficient de réflexion diffuse du matériau

$$I_{diff} = I_p K_d (\vec{n} \cdot \vec{s}) = I_p K_d \cos \theta$$

- Si la source est infiniment loin, θ est constant pour toute la surface éclairée.
- Sinon : facteur (empirique) d'atténuation de I_p en fonction de la distance D : $I_{p'} = \min(1, \frac{1}{a + bD + cD^2})$

8.1. Un peu de physique ...

Bibliographie

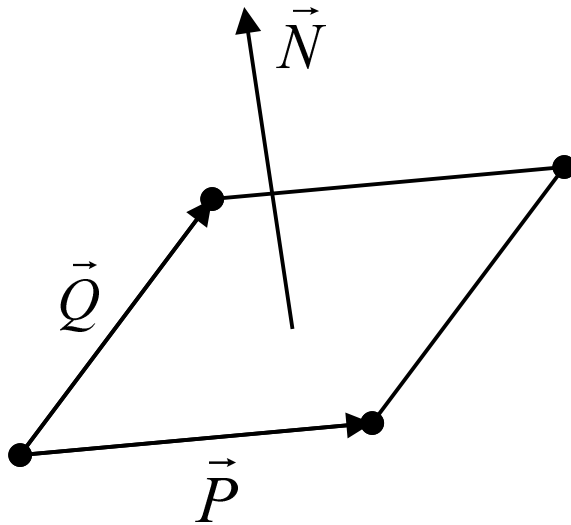
- P.Léna, A. Blanchard "Lumières, une introduction aux phénomènes optiques" InterEditions, 1990.
- Excellente introduction à la physique de la lumière. Niveau 1ère année DEUG ?
- M. Minnaert "The nature of light and color in the open air", Dover, 19??
- Moins mathématique que le précédent. Arcs en ciel, mirages, halos...
- B. Maitte "La lumière" collection "Points science", Seuil, 1981
- Introduction à l'histoire des théories concernant la lumière et plus largement au monde physique. Très clair.
- V. Ronchi "L'optique, science de la vision", Masson, 1966
- Un classique de l'histoire de l'optique, mais un peu vieilli (par rapport aux travaux de G. Simon par ex.)

8.2. Le coloriage à plat (*flat shading*) Bouknight 1970

- On utilise la loi de Lambert. L'intensité réfléchie est calculée pour chaque face et supposée constante pour toute la surface du polygone.
- L'approximation est valide si
 - la source est infiniment loin
 - l'observateur aussi, ou bien si la surface n'est pas réfléchissante

8.2. Le coloriage à plat (*flat shading*) Bouknight 1970

- Calcul de la normale à la face : On utilise le produit vectoriel (cf. chapitre 6)



$$\vec{N} = \vec{P} \wedge \vec{Q} \Rightarrow N = P \cdot Q \cdot \sin \theta$$

$$\begin{cases} x_N = y_P \cdot z_Q - z_P \cdot y_Q \\ y_N = z_P \cdot x_Q - x_P \cdot z_Q \\ z_N = x_P \cdot y_Q - y_P \cdot x_Q \end{cases}$$

- Les vecteurs N et S sont ensuite normalisés

8.2. Le coloriage à plat (*flat shading*) Reprise de l'algorithme

- Gestion des couleurs – attribution d'une couleur par objet
- Gestion des lumières – ce sont des lumières omnidirectionnelles (donc, des points 3D)
- Calcul des normales aux faces (lorsque l'objet est chargé)
- Ajout d'une valeur d'intensité lumineuse dans la classe face
- Modification de la procédure d'affichage pour prendre en compte ces modifications

8.2. Le coloriage à plat (*flat shading*)

Calcule des normales et intensités

```
class Face {
    Point3D normale;           // La normale à la face
    float intensite;          // L'intensité lumineuse de la face
    [...]

    public void calculeNormale() {
        if (normale == null) {
            Point3D P = sommets[1].point3D.moins(sommets[0].point3D);
            Point3D Q = sommets[2].point3D.moins(sommets[0].point3D);
            normale = P.vectoriel(Q);
            normale.normaliser();
        }
    }

    void calculeCouleur(Color CouleurObjet, Point3D lumieres[]) {
        intensite = 0.0f;
        for (int i=0;i<lumieres.length;i++) {
            Point3D L = lumieres[i].moins(sommets[0].point3D);
            L.normaliser();
            intensite += (L.scaire(normale)+1)/2; // Intensité entre 0 et 1
        }
        intensite /= lumieres.length;
    }
}
```

8.2. Le coloriage à plat (*flat shading*) Modification dans la méthode affichage

```
for (int i=0;i<objets.length;i++) {
    if (visibility) objets[i].calculeVisibilite(0);
    if (flat == true && lumieres != null)
        calculeCouleurs(lumieres);
    objets[i].projectionPerspective(R, theta, phi, D);
}

eviteClipping();

for (int i=0;i<objets.length;i++) {
    objets[i].calculeCoordonneesEcran(min, max, A, B);
    objets[i].affiche(imageGraphics, rendering, shading);
}
```

8.2. Le coloriage à plat (*flat shading*) Résultats

- Avec le fichier modifié de la venus

```
1000 90 -270 15
```

```
1
```

```
10 10 100
```

```
1
```

```
200 200 200
```

```
711
```

```
-0.814998 5.089999 0.640020
```

```
-0.784996 4.399999 0.900019
```

```
-0.934999 4.619999 0.400020
```

```
-0.684994 4.899999 1.300019
```

```
[...]
```

```
1396
```

```
2 1 0
```

```
3 0 1
```

```
4 3 1
```

```
5 3 4
```

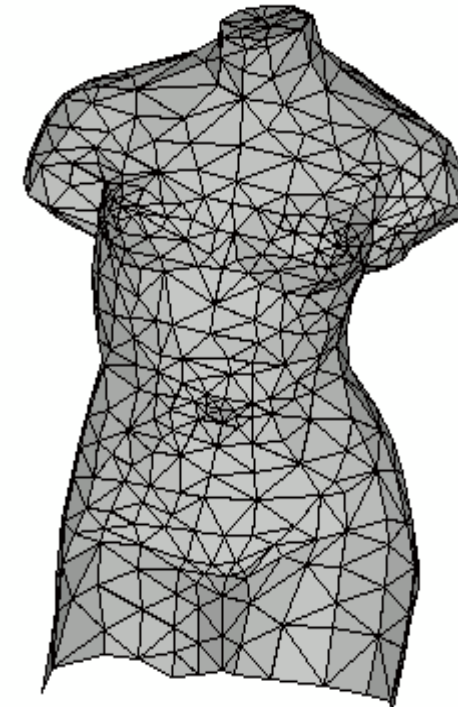
```
4 6 5
```

```
7 5 6
```

```
[...]
```

Une lumière en
(10, 10, 100)

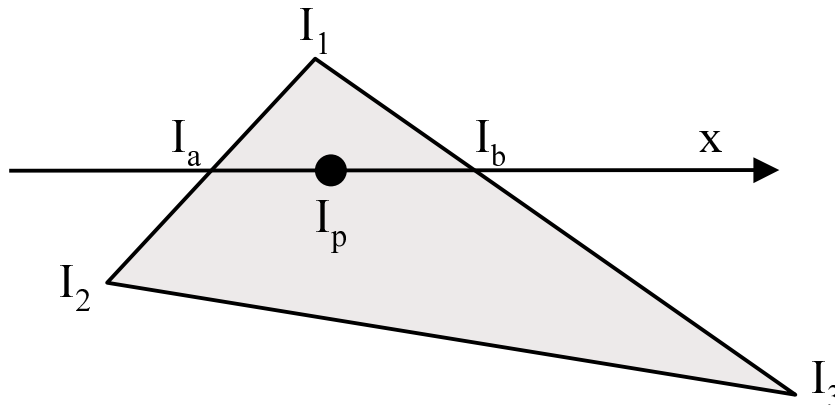
La couleur de
l'objet



8.3. Interpolation de Gouraud (*Gouraud shading*) 1971

- Principe : interpolation linéaire de l'intensité pour éliminer les discontinuités

Une technique similaire à l'interpolation de la profondeur pour le Z-Buffer :



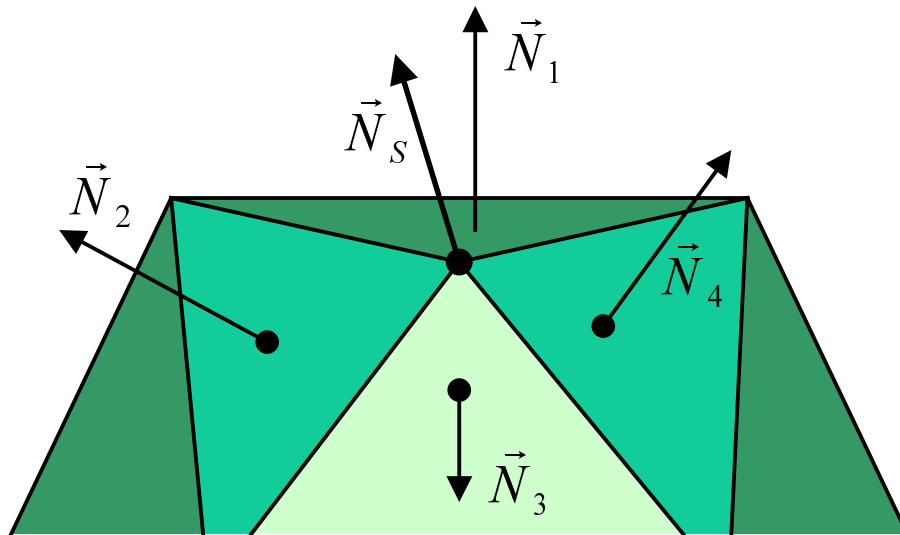
$$I_a = \frac{I_1(y_s - y_2) + I_2(y_s - y_1)}{y_1 - y_2}$$

$$I_b = \frac{I_1(y_s - y_3) + I_3(y_1 - y_s)}{y_1 - y_3}$$

$$I_p = \frac{I_b(x_b - x_p) + I_a(x_p - x_a)}{x_b - x_a}$$

8.3. Interpolation de Gouraud (*Gouraud shading*) 1971

- L'intensité d'un sommet est obtenue grâce à sa normale calculée à partir des normales de ces faces adjacentes



$$\vec{N}_s = \frac{\vec{N}_1 + \vec{N}_2 + \vec{N}_3 + \vec{N}_4}{\|\vec{N}_1 + \vec{N}_2 + \vec{N}_3 + \vec{N}_4\|}$$

8.3. Interpolation de Gouraud Reprise de l'algorithme

- Calcul des normales aux sommets
- Calcul des intensités lumineuses aux sommets
- Modification de la procédure d'affichage pour prendre en compte ces modifications

8.3. Interpolation de Gouraud

Calcule des normales et intensités

```
class Sommet {
    Point3D normale;           // La normale au sommet
    float intensite;          // L'intensite l'umineuse au sommet
    [...]

    void calculeCouleur(Color CouleurObjet, Point3D lumieres[]) {
        intensite=0.0f;
        for (int i=0;i<lumieres.length;i++) {
            Point3D L = lumieres[i].moins(point3D);
            L.normaliser();

            intensite += (L.scalaire(normale)+1)/2; // Intensité entre 0 et 1
        }

        intensite /= lumieres.length;
    }
}
```

8.3. Interpolation de Gouraud

Calcule des normales et intensités

```
class Face {
    [...]
    public void calculeNormales() {
        // Les normales aux faces
        for (int i=0;i<faces.length;i++) {
            faces[i].calculeNormale();
        }

        // Et les normales aux sommets = la somme des normales des faces communes
        for (int j=0;j<faces.length;j++) {
            // Ajout du vecteur normal à la face
            for (int i=0;i<faces[j].sommets.length;i++) {
                faces[j].sommets[i].normale.x += faces[j].normale.x;
                faces[j].sommets[i].normale.y += faces[j].normale.y;
                faces[j].sommets[i].normale.z += faces[j].normale.z;
            }
        }

        // Normalisation des normales aux sommets
        for (int i=0;i<sommets.length;i++) {
            sommets[i].normale.normaliser();
        }
    }
}
```

8.3. Interpolation de Gouraud

Modifications dans la méthode affichage

```
// Interpolation des couleurs entre chacun des sommets
// Calcul de l'intensité du point balayé
double ymin = sommets[0].point2D.y;
double ymax = ymin;
for (int i=1;i<sommets.length;i++) {
    if (sommets[i].point2D.y < ymin) ymin = sommets[i].point2D.y;
    if (sommets[i].point2D.y > ymax) ymax = sommets[i].point2D.y;
}

// Balayage de ymin à ymax
for (int j=(int)ymin;j<=(int)ymax;j++) {
    // Chercher les intersections avec les arêtes du polygone
    // intersections de deux droites :
    //   - la droite de balayage
    //   - la droite liant deux sommets consécutifs du polygone
    // On ne traite que les polygones convexes
    // --> seulement deux intersections possibles par ligne balayée
    Point2D Inter2D[] = new Point2D[2];
    float InterIntensite[] = new float[2];
```

8.3. Interpolation de Gouraud

Modifications dans la méthode affichage

```
int inter=0;
for (int i=0;i<sommets.length&&inter<2;i++) {
    // Calcul de l'équation de la droite passant par les deux sommets
    //  $y = ax+b \implies a = (By-Ay)/(Bx-Ax)$  et  $b = Ay-a.Ax$ 
    Sommet S1 = sommets[i];
    Sommet S2 = sommets[(i+1)%sommets.length];
    double A = (S2.point2D.y-S1.point2D.y)/(S2.point2D.x-S1.point2D.x);
    double B = S1.point2D.y-A*S1.point2D.x;

    Point2D I = new Point2D(((double)j-B)/A, (double)j);
    // Le point I est-il sur l'arrête ?
    // Oui, si  $S1I.S1S2 \geq 0$  et  $IS2.S1S2 \geq 0$ 
    Point2D S1I = I.moins(S1.point2D);
    Point2D IS2 = S2.point2D.moins(I);
    Point2D S1S2 = S2.point2D.moins(S1.point2D);

    if (S1I.scalaire(S1S2)>=0 && IS2.scalaire(S1S2)>=0) {
        Inter2D[inter] = new Point2D(I);
        InterIntensite[inter] = S1.intensite+(S2.intensite-
            S1.intensite)*(float)(S1I.norme()/S1S2.norme());

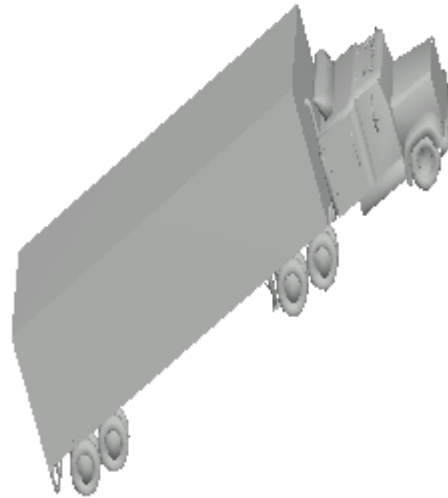
        inter++;
    }
}
```

8.3. Interpolation de Gouraud

Modifications dans la méthode affichage

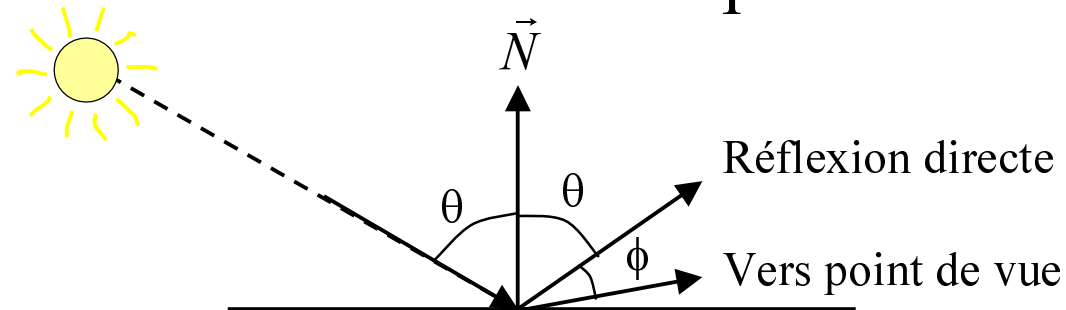
```
if (inter == 2) {  
    // Classement des points par X croissant  
    if (Inter2D[0].x > Inter2D[1].x) {  
        Point2D temp = new Point2D(Inter2D[0]);  
        float tempInt = InterIntensite[0];  
  
        Inter2D[0] = new Point2D(Inter2D[1]); InterIntensite[0] = InterIntensite[1];  
        Inter2D[1] = new Point2D(temp); InterIntensite[1] = tempInt;  
    }  
  
    // On a les deux points d'intersection triés  
    // Maintenant, on balaye  
    for (int i=(int)Inter2D[0].x;i<(int)Inter2D[1].x;i++) {  
        float intensitePoint = (InterIntensite[0]*((int)Inter2D[1].x-i)+  
                                InterIntensite[1]*(i-(int)Inter2D[0].x))  
                                /((int)Inter2D[1].x-(int)Inter2D[0].x);  
        gr.setColor(new Color((int)(couleur.getRed()*intensitePoint),  
                               (int)(couleur.getGreen()*intensitePoint),  
                               (int)(couleur.getBlue()*intensitePoint)));  
        gr.drawLine(i, j, i, j);  
    }  
}  
}
```

8.3. Interpolation de Gouraud Résultats



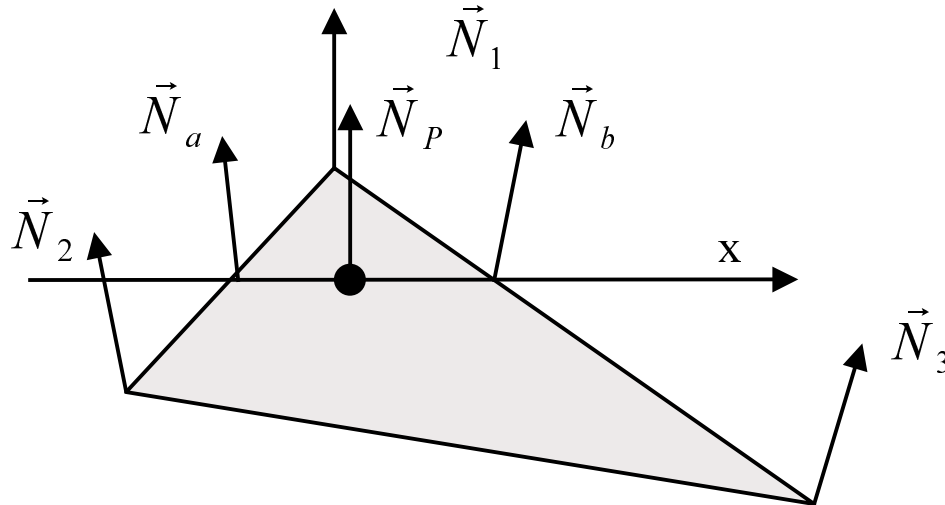
8.4. Modèle de Phong Bui-Tuong (*Phong Shading*) 1975

- Le principe :
 - Calcul d'une réflexion spéculaire en chaque point de la face, variable selon le matériau
 - Donne un objet avec une réflexion spéculaire mais pas entièrement, d'où l'aspect de brillance locale
- On doit ici tenir compte de la position de l'observateur et de la normale au point calculé



8.4. Modèle de Phong Bui-Tuong (*Phong Shading*) 1975

- Pour déterminer θ localement, on interpole les normales au lieu des intensités



$$\vec{N}_a = \frac{(y_1 - y_s)\vec{N}_1 + (y_s - y_2)\vec{N}_2}{y_1 - y_2}$$

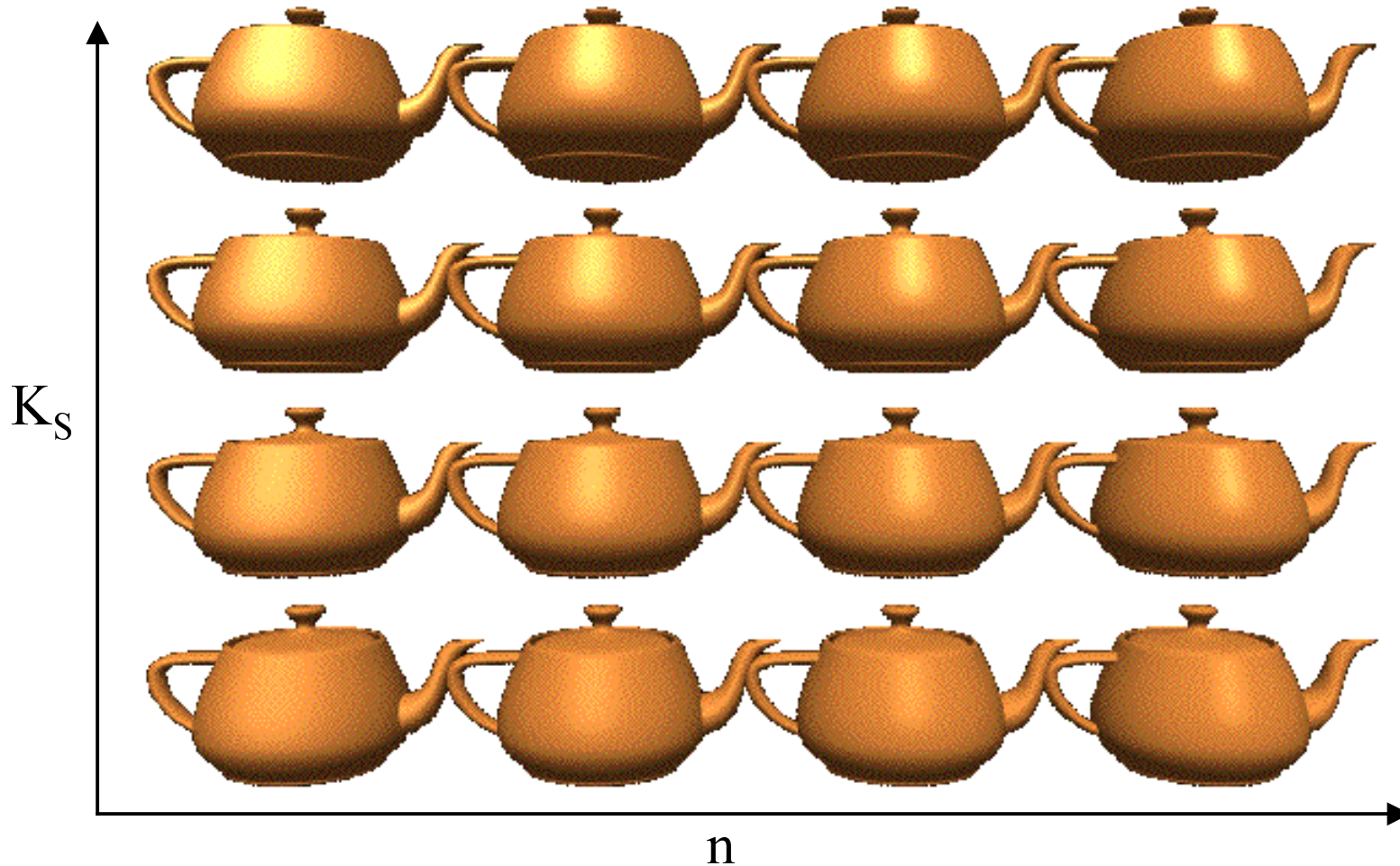
$$\vec{N}_b = \frac{(y_1 - y_s)\vec{N}_1 + (y_s - y_3)\vec{N}_3}{y_1 - y_3}$$

$$\vec{N}_P = \frac{(x_P - x_a)\vec{N}_a + (x_b - x_P)\vec{N}_b}{x_b - x_a}$$

- Modèle (empirique) retenu :

$$I_{spec} = I_P K_S \cos^n \phi \Rightarrow I_{total} = I_{diff} + I_{spec} = I_P (K_d \cos \theta + K_S \cos^n \phi)$$

8.4. Modèle de Phong Bui-Tuong (*Phong Shading*) 1975



8.4. Modèle de Phong Bui-Tuong (*Phong Shading*) 1975

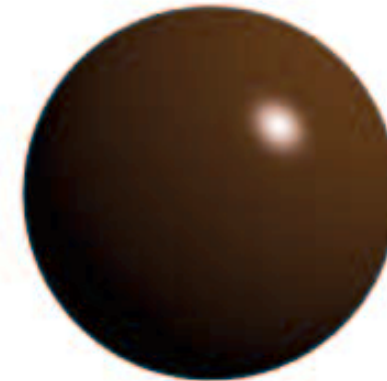
- Exemple avec POV
 - Option de rendu :
`finish { phong Ks phong_size n }` avec $0 \leq K_s \leq 1$ et $1 \leq n \leq 256$
- Résultat sur une sphère



$K_s=1, n=5$



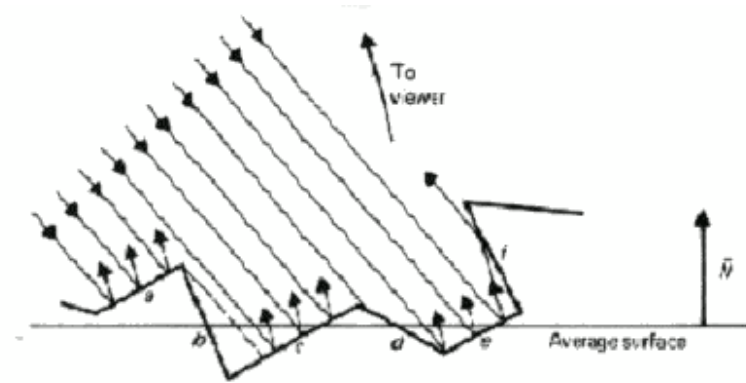
$K_s=1, n=10$



$K_s=1, n=50$

8.5. Modèles de micro-facettes

- Modèle de Davies (1954) .
 - Pas d'interaction entre les micro-facettes + distribution des hauteurs selon une loi normale
- Modèle de Torrance & Sparrow (1966) :
 - Facettes "en V", réfléchissantes, orientées de manière aléatoire selon une loi de Beckmann



8.5. Modèles de micro-facettes

- Le modèle de POV :
`finish {specular Ks roughness rough }`
- Exemple avec la sphère :



$K_s=1$
rough = 0.1



$K_s=1$
rough = 0.5

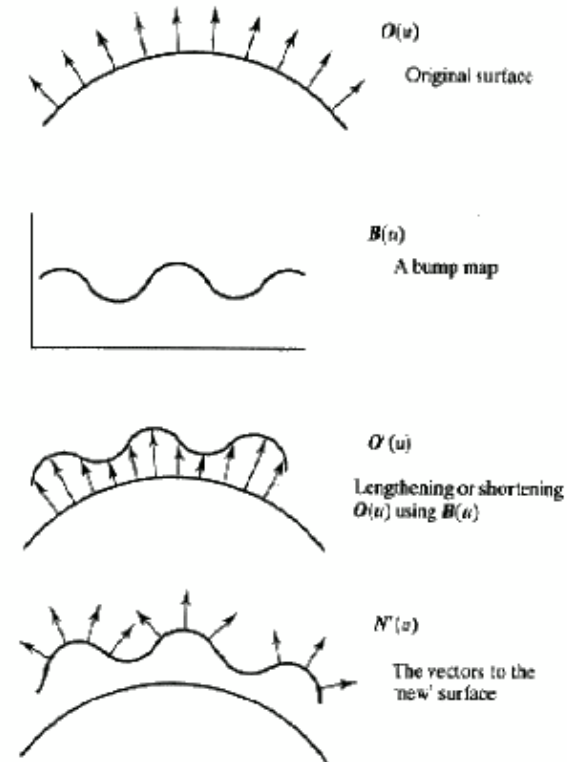


$K_s=1$
rough = 1.0

8.5. Modèles de micro-facettes

Le "bump mapping"

- Perturber la normale à la surface
 → Astuce pour donner un relief aux objets.
- Plusieurs techniques :
 - Le plus simple : perturbation aléatoire
 - Le plus général : un tableau indiquant pour chaque point de la surface comment se fait la perturbation



8.5. Modèles de micro-facettes

Le "bump mapping"

- Le modèle de POV :
`finish {bumps hauteur scale largeur }`
- Exemple avec la sphère :



bumps = 0.3
scale = 0.2



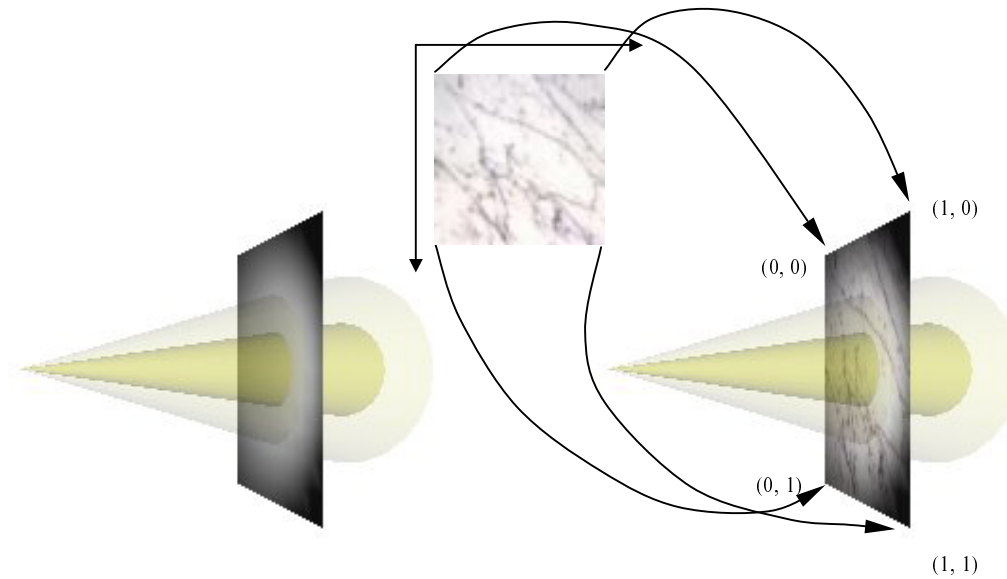
bumps = 0.5
scale = 0.1



bumps = 1
scale = 0.05

8.6. Textures

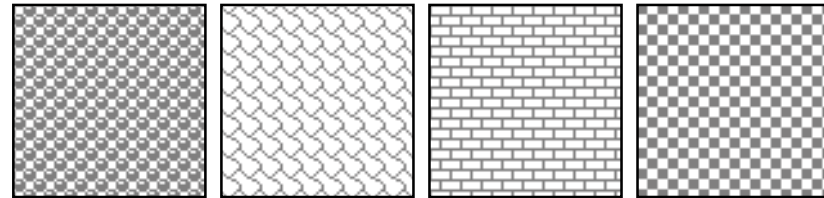
- Les algorithmes précédents ne retirent pas complètement le caractère "artificiel" du coloriage.
- Une solution souvent employée : le plaqué de textures sur les faces



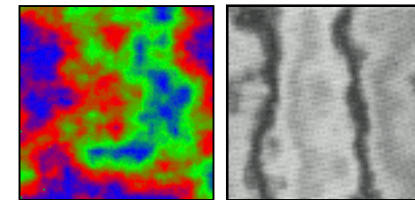
8.6. Textures

- Divers types de textures :

- "géométriques"
→ forme répétitive



- "synthétiques"
→ génération aléatoire



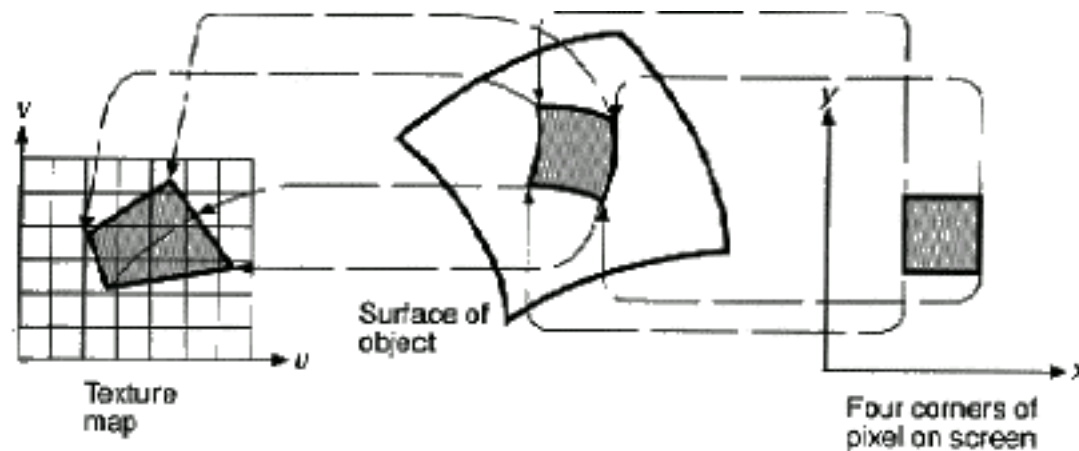
- "réalistes"
→ photographie numérisée



8.6. Textures

Projection de la texture

- Textures "géométriques" :
 - la texture est décrite par une matrice P de taille 16×16
 - Pour chaque pixel (x,y) du polygone, la couleur de base est déterminée par $P[x \bmod 16, y \bmod 16]$
- Les autres : problème de projection



[FOLEY] p. 743

8.6. Textures

Projection de la texture

- Projections cylindriques et sphériques
 (voir Heckbert "Survey of texture mapping" CG&A Novembre 1986, pp. 56-67)

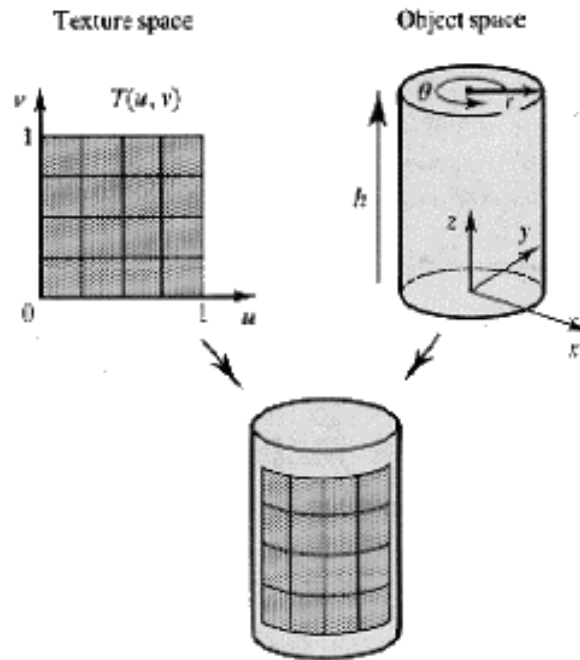


Figure 6.1 The cylindrical mapping.

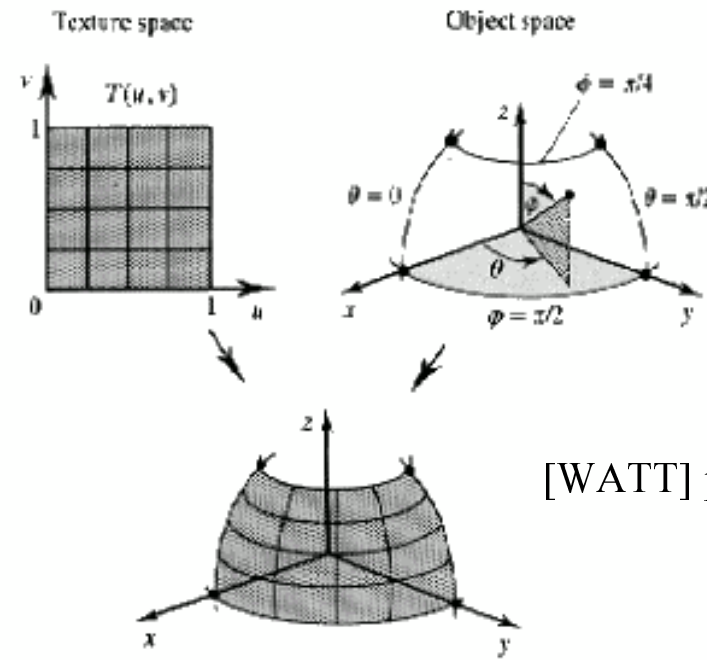


Figure 6.2 The spherical mapping.

[WATT] p. 180

8.6. Textures

Projection de la texture

- Exemple en POV

```
background { rgb <1, 1, 1> }
```

```
camera {  
  location <0,0,-3>  
  look_at <0,0,0>  
}
```

```
light_source { <1000,1000,-1000> rgb <1,1,1> }  
light_source { <0,0,-1000> rgb <1,1,1> }
```

```
sphere {  
  <0, 0, 0>, 1  
  texture {  
    pigment {  
      image_map { gif "planisphere.gif" map_type 1 }  
    }  
  }  
}
```

