Shading: Tedious reality

- Properly determining the right color is really hard
- Look around the room.
  - Each light source has different characteristics
  - Trillions of photons are pouring out every second
- These photons can:
  - interact with the atmosphere, or with things in the atmosphere
  - strike a surface and
    - be absorbed
    - be reflected
    - be refracted
    - cause fluorescence or phosphorescence
  - at some microscopic level (very important for photons) all surfaces are really bumpy
  - also, everything depends on wavelength

Local illumination models

- Local illumination:
  - consider only the light source(s) and a surface point
- Global illumination:
  - also take all the other surfaces into account

Phong illumination model

- We’re going to build up to an approximation of reality called the Phong illumination model
- It has the following characteristics:
  - a local illumination model
  - not physically based
  - gives a first-order approximation to physical light reflection
  - very fast
  - widely used

Ambient component

- Let’s make the color at least dependent on the overall quantity of light available in the scene:
  \[ I = k_a I_a \]
- \( k_a \) is the ambient reflection coefficient
  - really the reflectance of ambient light
  - “ambient” light is assumed to be equal in all directions
- \( I_a \) is the ambient intensity

- Physically, what is “ambient” light?
  Light that has diffusely reflected several times from various surfaces, particles in the air, etc., so that it doesn’t favor any particular direction any more.
Wavelength dependence

- Really, $k_a$ and $I_a$ are functions over all wavelengths $\lambda$.
- **Ideally**, we would do the calculation on these functions:
  $$I(\lambda) = k_a(\lambda)I_a(\lambda)$$
- then find good RGB values to represent the spectrum $I(\lambda)$.
- **In practice**, $k_a$ and $I_a$ are represented as RGB triples.
  - the computation is performed on each color channel separately.
- **Problem:**
  - in reality, use red light to illuminate a green apple => it looks brownish.
  - with RGB model, $(0,1,0) \cdot (1,0,0) = (0,0,0)$ it becomes black!

Diffuse vs. ambient reflection

- Let’s examine the ambient shading model:
  - objects have different colors
  - we can control the overall light intensity
- So far, objects are uniformly lit.
  - not the way things really appear
  - looks totally flat
  - in reality, light sources are directional.
- Diffuse, or Lambertian reflection will allow reflected intensity to vary with the direction of the light.

Diffuse reflectors

- Diffuse reflection occurs from dull, matte surfaces, like chalk.
- These diffuse or Lambertian reflectors reradiate light equally in all directions.
- A rough surface can be modeled to have lots of tiny microfacets.
- The microfacets have the effect of distributing light rays in all directions.
- **Note:**
  - Light may actually penetrate the surface, bounce around, and then reflect back out.
  - Accounts for colorization of diffusely reflected light by plastics.

Diffuse reflectors, cont.

- The reflected intensity from a diffuse surface does not depend on the direction of the viewer.
- The incoming light, though, does depend on the direction of the light source.
- **Q:** Why is Finland colder than Sahara? Why is winter cold?
  - Sahara, at the equator, faces the sun almost perpendicularly.
  - In the winter (Dec-Feb) earth's axis aims the northern hemisphere away from the sun, the rays hit surface with more slant.
**Diffuse component**

- The incoming energy is proportional to \( \cos \theta \), giving the diffuse reflection equations:
  \[
  I = k_d I_d + k_a I_a \cos \theta
  = k_d I_d + k_a I_a (n \cdot l),
  \]
- where:
  - \( k_d \) is the **diffuse reflection coefficient**
  - \( I_d \) is the (diffuse) intensity of the light source
  - \( n \) is the normal to the surface (unit vector)
  - \( l \) is the direction to the light source (unit vector)
  - \((x)_+ \) means \( \max \{0, x\} \)

\[
\mathbf{a} \cdot \mathbf{b} = |\mathbf{a}||\mathbf{b}| \cos \theta
\]

**Specular reflection**

- **Specular reflection** accounts for the highlight that you see on some objects

  - It is particularly important for smooth, shiny surfaces, such as:
    - metal
    - polished stone
    - plastics
    - apples on a supermarket
  
- The color is often determined solely by the color of the light
  - corresponds to absence of internal reflections

**Specular reflection derivation**

- Specular reflection depends on the viewing direction \( \mathbf{v} \)
- For a perfect mirror reflector, light is reflected about \( \mathbf{n} \), so
  \[
  I = \begin{cases} 
  I_l & \text{if } \mathbf{v} = \mathbf{r} \\
  0 & \text{otherwise}
  \end{cases}
  \]
- For a near-perfect reflector, you might expect the highlight to fall off quickly with increasing angle \( \phi \)
- Also known as:
  - “rough specular” reflection
  - “directional diffuse” reflection
  - “glossy” reflection

**Derivation, cont.**

- One way to model near-perfect reflector is to take \( \cos \phi = (\mathbf{r} \cdot \mathbf{v})/(|\mathbf{r}||\mathbf{v}|) \), raised to a power \( n_s \)

\[
I = \left( \cos \phi \right)^{n_s}
\]

- As \( n_s \) gets larger,
  - the dropoff becomes {more,less} gradual
  - gives a {larger,smaller} highlight
  - simulates a {more,less} glossy surface

\[
\cos \phi - \frac{\pi}{2} 0 \frac{\pi}{2}
\]

\[
\phi
\]

\[
-\frac{\pi}{2} 0 \frac{\pi}{2}
\]
Fresnel effect

- Reflectance really depends on the viewing angle
  - slanted angles reflect more
- Try it out
  - take a piece of paper, should be pretty diffuse
  - look over it almost parallel to the surface
    - if you look at an object that’s bright enough, the sheet of paper almost acts like a mirror!

Intensity drop-off with distance

- The intensity of a point light source drops off with its distance squared
- We can incorporate this effect by multiplying $I_i$ by $1/d^2$
- Sometimes, this distance-squared drop-off is considered too “harsh”, equation

$$\frac{1}{a + bd + cd^2}$$

with user-supplied constants for $a$, $b$, and $c$ gives more control

Full Phong model

- Since light is additive, we can handle multiple lights by taking the sum over every light
- Our equation is now:

$$I = k_a I_a + \sum_i f(d_i) I_i \left( k_d \left( n \cdot l \right)_+ + k_s \left( v \cdot r \right)_+ \right)$$

- This is the Phong illumination model
- Which quantities are
  - spatial vectors? $n$, $l_i$, $v$, $r$
  - RGB triples? $l_a$, $l_i$, $k_a$, $k_s$
  - scalars? $d_i$, $f(d_i)$, $n_s$

Accelerating lighting

- The reflection vector can be approximated by a halfway vector which is cheaper to calculate
- The angle between $n$ and $h$ is half of the angle between $r$ and $v$, so $n \cdot h = r \cdot v$
- However, as $r \cdot v$ is an approximation not based on physics, this one does as well
  - can compensate by adjusting $n_s$
- Further speedup can be obtained by treating the viewer as infinitely far away
  - $v$ becomes $(0, 0, 1)$' since lighting is done in camera coordinates

$$r = -l + 2 \frac{l \cdot n}{n \cdot n} n$$

$$h = l + v$$
OpenGL version

- OpenGL gives more control than the traditional Phong model
- Each light gets its own ambient, diffuse, and specular coefficients
  - specular term is accelerated with halfway vector
- The surface can also have an emissive term

\[ I = I_e + k_a I_a + \sum_i \left[ k_d I_d a_i + f(d_i) (k_d I_d \cdot \mathbf{n} \cdot \mathbf{l}_i) + k_s I_s (\mathbf{n} \cdot \mathbf{h})^\ast \mathbf{n}_i \right] \]

---

OpenGL functions

- \textbf{glMaterial} (face, pname, params)
  - int or float, vector or scalar version
  - face := \{GL_FRONT | GL_BACK | GL_FRONT_AND_BACK\}
  - pname := \{GL_AMBIENT | GL_DIFFUSE | GL_SPECULAR | GL_EMISSION | GL_SHININESS | GL_AMBIENT_AND_DIFFUSE | GL_COLOR_INDEXES\}

- \textbf{glLightModel} (pname, params)
  - int or float vector
  - pname := \{GL_LIGHT_MODEL_AMBIENT | GL_LIGHT_MODEL_LOCAL_VIEWER | GL_LIGHT_MODEL_TWO_SIDE\}
  - the global ambient light color (default (.2,.2,.2,1))
  - using real viewer location in specularity calculations (def: 0 = fast)
  - two-sided lighting calculations (def: 0 = one-sided)

- \textbf{glColorMaterial} (face, pname)
  - make the material to track the current color
  - \textbf{glEnable/glDisable} (GL_COLOR_MATERIAL)

---

Use unit normals

- Lighting works only if you give also normals, and OpenGL assumes that they are vectors of length 1.0
- OpenGL transforms normals differently from vertices, by the \textit{inverse transpose modelview} matrix. Any enlarging or shrinking of vertices during the modelview transformation also changes the length of normals.

- Two ways to avoid wrong normals (and hence wrong colors)
  - Use \textbf{glEnable(GL_NORMALIZE)}, which forces OpenGL to normalize transformed normals to be of unit length before using them.
  - It also slows OpenGL's vertex processing speed.
  - Adjust the normal vectors passed to OpenGL so that after the inverse transpose modelview transformation, the resulting normal will become a unit vector.
  - For example, if the modeling transformation scales the object by three, we could correct for the corresponding thirding effect on the transformed normals by pre-multiplying each normal component by three.
Computing normal vectors: triangle
- Use counterclockwise (CCW) ordering
- Take the cross product of two edges

\[ n = (p_1 - p_0) \times (p_2 - p_0) \]

Normals: implicit surface
- Implicit surface: \( f(x, y, z) = 0 \)
- Gradient vector is aligned with surface normal

\[ n \parallel \nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \\ \frac{\partial f}{\partial z} \end{bmatrix} \]

Example: unit sphere
\[ f(x, y, z) = x^2 + y^2 + z^2 - 1 = 0 \]
\[ \nabla f = \begin{bmatrix} 2x \\ 2y \\ 2z \end{bmatrix} \]
\[ n = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \]

Normals: parametric surface
- A surface parameterized by \( u,v: \)
\[ f(u,v) = \begin{bmatrix} x(u,v) \\ y(u,v) \\ z(u,v) \end{bmatrix} \]
- Take two tangent vectors
\[ \frac{\partial f}{\partial u}, \frac{\partial f}{\partial v} \]
- How do we get the normal? \( \text{cross the tangents} \)

Choosing reflectance parameters
- How would I model…
  - blue plastic? \( \text{blue diffuse, white specular} \)
  - polished copper? \( \text{orange diffuse, yellow specular, large shininess coeff} \)
  - lunar dust? \( \text{gray diffuse, no specular} \)
  - light bulb? \( \text{yellowish white emitting light, maybe white specular and some shininess for the glass} \)
Some shiny examples

# Aluminum
ambient 0.3 0.3 0.3
diffuse 0.3 0.3 0.5
specular 0.7 0.7 0.8
shininess 13

# Copper
ambient 0.26 0.26 0.26
diffuse 0.3 0.11 0.0
specular 0.75 0.33 0.0
shininess 10

# Gold
ambient 0.4 0.4 0.4
diffuse 0.22 0.15 0.0
specular 0.71 0.7 0.56
shininess 20

# Metallic Purple
ambient 0.17 0.17 0.17
diffuse 0.1 0.03 0.22
specular 0.64 0.0 0.98
shininess 20

# Metallic Red
ambient 0.15 0.15 0.15
diffuse 0.27 0.0 0.0
specular 0.61 0.13 0.18
shininess 20

# Plastic Blue
ambient 0.1 0.1 0.1
diffuse 0.20 0.2 0.71
specular 0.83 0.83 0.83
shininess 15

Some more materials (teapots.py)

- amb, diff, spec, shininess / 128
  - emerald (0.0215, 0.1745, 0.0215, 0.07568, 0.61424, 0.07568, 0.633, 0.727811, 0.633, 0.6)
  - jade (0.135, 0.2225, 0.1575, 0.54, 0.89, 0.63, 0.3162, 0.3162, 0.3162, 0.1)
  - obsidian (0.05375, 0.05, 0.06625, 0.18275, 0.17, 0.22525, 0.332741, 0.328634, 0.346435, 0.3)
  - pearl (0.25, 0.20725, 0.20725, 1, 0.829, 0.829, 0.29665, 0.29665, 0.29665, 0.088)
  - ruby (0.1745, 0.01175, 0.01175, 0.61424, 0.04136, 0.04136, 0.727811, 0.626959, 0.626959, 0.6)
  - turquoise (0.1, 0.18725, 0.1745, 0.396, 0.74151, 0.69102, 0.297254, 0.30829, 0.306678, 0.1)
  - ...

Blender materials

- Shape of example object (plane, sphere, cube)
- Dark/bright background selection
- Color space
- Specular color
- Diffuse color
- Intensity of specular spot
- How much of the global ambient is applied
- "Specular transparency"
- "Hardness" or size of specular spot
- Reflectance of light source
- Opaqueness (use with Ztransp)
- Light emission (diffuse)

Simple illumination models

- Point source
- Directional light
Spot light

- Direction
  - Cutoff angle
  - Intensity attenuation
  - Attenuation exponent

OpenGL functions

- `glLight(light, pname, params)`
  - `light` := `GL_LIGHTi`, where 0 ≤ i ≤ `GL_MAX_LIGHTS`
  - `pname` := `{GL_AMBIENT | GL_DIFFUSE | GL_SPECULAR | GL_POSITION | GL_SPOT_DIRECTION | GL_SPOT_EXPONENT | GL_SPOT_CUTOFF | GL_CONSTANT_ATTENUATION | GL_LINEAR_ATTENUATION | GL_QUADRATIC_ATTENUATION}

- `glEnable/glDisable()`
  - `GL_LIGHTING`
  - `GL_LIGHTi`

Three point lighting

- A classical lighting scheme for objects and characters
- Key light illuminates target
- Back light emphasizes the silhouettes of target
- Fill light fills in shadows and reduces overall contrast
- Background may have different lighting

Note about colors vs. shading

- In OpenGL, you can either use the current colors or current materials, not both
  - except by using `glColorMaterial` so material tracks current color
- If `GL_LIGHTING` has been enabled, the current colors are ignored
- So that you can see anything, you need
  - valid, enabled lights
  - non-zero current material properties
  - valid normal vectors
  - and the object within view frustum :-(
**Simple participating medium**

- Attenuation and colorization due to fog
  - depth cueing
- Blend fog color with the object color
  - depends on the viewing distance
  - modes: GL_LINEAR, GL_EXP, GL_EXP2

\[
gl\text{Fog}(\text{pname, params})
\]
- int or float, vector or scalar version
- \text{pname:} = \{GL\_FOG\_MODE, GL\_FOG\_DENSITY, GL\_FOG\_START, GL\_FOG\_END, GL\_FOG\_INDEX, GL\_FOG\_COLOR\}

\[
gl\text{Enable/Disable(}GL\_FOG)\]

---

**Modeling vs. measuring**

- The Phong model is just a simplification
  - a function that easy to evaluate
  - still a reasonable approximation of reflectance
- If you want high-quality realistic reflectance
  - it’s better to simply measure it
  - shine light in
  - see how much comes out

---

**BRDF**

- Bidirectional Reflectance Density Function
  - the "real" reflectance
  - given a 2D point on surface,
    - it’s a 4D function
    - 2D: direction of light coming in
    - 2D: direction of light going out
    - how much of the incoming light goes there (frequency dependent)
- Isotropic/anisotropic
  - if you can rotate the material around normal without reflectance changing it’s isotropic
  - otherwise anisotropic
- Picture shows the BRDF of the Phong model

---

**BSSRDF**

- Bidirectional surface scattering distribution function
  - the light can travel in the matter before going out
  - and often does
  - 8D: 2D in direction
    - 2D in location
    - 2D out location
    - 2D out direction

- Picture shows the BSSRDF of milk, skim milk, and whole milk
BRDF vs. BSSRDF

VRML 97

- VRML = Virtual Reality Markup Language
- A simple text language for describing 3D shapes and interactive environments
  - we'll concentrate on static modeling
- Viewing VRML (*.wrl) files
  - VRML helper application
  - HTML-browser plugin
  - we'll write our own

VRML 97

- VRML = Virtual Reality Markup Language
- A simple text language for describing 3D shapes and interactive environments
  - we'll concentrate on static modeling
- Viewing VRML (*.wrl) files
  - VRML helper application
  - HTML-browser plugin
  - we'll write our own

VRML 97

- VRML = Virtual Reality Markup Language
- A simple text language for describing 3D shapes and interactive environments
  - we'll concentrate on static modeling
- Viewing VRML (*.wrl) files
  - VRML helper application
  - HTML-browser plugin
  - we'll write our own

VRML 97

- VRML = Virtual Reality Markup Language
- A simple text language for describing 3D shapes and interactive environments
  - we'll concentrate on static modeling
- Viewing VRML (*.wrl) files
  - VRML helper application
  - HTML-browser plugin
  - we'll write our own

File structure

- File header
  - version, character set
- Comments
- Nodes
  - pieces of scene information
- Fields
  - node attributes you can change
- Values
  - attribute values
  - a default value is used if the field is not set

XML & X3D

- XML = eXtended Markup Language
  - everything is enclosed between matching structured tags
  - tags make out hierarchies
  - they make documents machine readable
- X3D
  - the next generation VRML
  - encodes in XML
  - many additions

XML & X3D

- XML = eXtended Markup Language
  - everything is enclosed between matching structured tags
  - tags make out hierarchies
  - they make documents machine readable
- X3D
  - the next generation VRML
  - encodes in XML
  - many additions

XML & X3D

- XML = eXtended Markup Language
  - everything is enclosed between matching structured tags
  - tags make out hierarchies
  - they make documents machine readable
- X3D
  - the next generation VRML
  - encodes in XML
  - many additions

XML & X3D

- XML = eXtended Markup Language
  - everything is enclosed between matching structured tags
  - tags make out hierarchies
  - they make documents machine readable
- X3D
  - the next generation VRML
  - encodes in XML
  - many additions
Shapes

- Building blocks of VRML world
- Fields
  - appearance Node (color and texture)
  - geometry Node (form or structure)
- The geometry can be, e.g., Box, Cone, Cylinder, Sphere, Text

Transform

- Transform creates a new coordinate system that is positioned, oriented, and scaled with respect to the parent coordinates
- Fields (in order of application)
  - scale x y z
  - rotation x y z angle
    # xyz is the axis, angle is radians
  - translation x y z
  - children is a list of nodes

Example

- A red box at \([1,3],[-.5,.5],[-.5,.5]\)

Appearance

- The Appearance node can contain material and/or texture properties
- The Material node has these fields (with the given defaults)
Grouping nodes

- Group shows every child
- Switch shows only one child
  - or none if the choice is negative
- Transform sets an own coordinate system and shows all children
- Billboard always rotates the group toward camera
- Inline reads in and includes a given file
- Anchor provides a hyperlink if one of the children is clicked

```
Group {
  children [ ... ]
}
Switch {
  whichChoice -1
  children [ ... ]
}
Billboard {
  axisOfRotation ... 
  children [ ... ]
}
Inline {
  url "blah.wrl"
}
Anchor {
  url "blah.wrl"
  children [ ... ]
}
```

Naming nodes

- You can define a new name for a node and later use it

```
DEF RedColor Material {
  diffuseColor 1 0 0
}
...
Appearance {
  material USE RedColor
}
```

Building new shapes

- Coordinate contains vertex coordinates
  
  ```
  Coordinate {
    point [ 
      1 4 1.5,
      2.1 1 3,
      ...
    ]
  }
  <Coordinate point="1 4 1.5,
  2.1 1 3, ..."/>
  PointSet {
    coord Coordinate {
      point [ ... ]
    }
  }
  <PointSet>
  <Coordinate point="..."/>
  </PointSet>
  ```

- The coordinates can be used in the coord field of geometry nodes
  
  ```
  PointSet
  IndexedLineSet
  IndexedFaceSet
  ```

IndexedFaceSet

- A surface is created by creating faces (polygons)
  - defined by coordinates
  - indexed by coordIndex
  - Indices start with 0
  - Faces are separated by using index -1

```
IndexedFaceSet {
  coord Coordinate {
    point [ ... ]
  }
  coordIndex [ ... ]
}
```

```
<IndexedFaceSet
  coordIndex="0,1,2,-1,...">
  <Coordinate point="..."/>
  <TextureCoordinate point="..."/>
</IndexedFaceSet>
```
Controlling the viewpoint

- The first viewpoint in the file is the entry point
- Viewpoints can be transformed using Transform node

```xml
<Viewpoint
description="Hey pepperoni"
orientation="1 0 0 -0.758"
position="0 2 2"/>
```

Scene graphs M3G (JSR-184)

- Really a tree, not a graph
  - every node has a unique parent
  - the tree encodes structure
  - the actual data (vertices, colors, textures, animation data, …) can be shared
- World is the root
  - a special case of a group
- Other nodes
  - camera
  - light
  - mesh
  - sprite
- Nodes also encode relative transformations
  - and inherited alpha factor for transparency effects

Meshes in M3G

- Mesh is a
  - collection of vertices with properties
  - submeshes
    - that index to VertexBuffer to create triangle strips
    - that can have their own appearances (materials, textures, …)

Appearance for Meshes & Sprites

- alpha / depth tests & masking, blending modes
- culling, winding, shading, two-sided lighting, depth offset
- materials, fog, textures very much like in OpenGL
Display lists

- Typically, OpenGL renders in **immediate mode**
  - aka direct mode
- For **retained mode** rendering you need to create a display list
- Pros:
  - define object once, store the display list on server
  - need to only send a single function call from client to server
  - can be faster to execute
    (no need to execute other code than OpenGL)
- Cons:
  - takes up memory
  - overhead to create a display list

How to use

- Create a list, and call it
  - `glNewList(1, GL_COMPILE)`
  - `myFuncThatCallsOpenGL()`
  - `glEndList()`
  - `gllCallList(1)`
- Can also execute immediately with
  `GL_COMPILE_AND_EXECUTE`
- Can make hierarchical
  - `glNewList(2, GL_COMPILE)`
    - `...`  
    - `gllCallList(1)`
    - `...`  
    - `glEndList()`
- Cannot start a new list before the last one has ended

Other DL functions

- `dls = glGenLists(i)`
  - allocates `i` display list indices, they run continuously starting from `dls` (which is non-zero)

- `glIsList(i)`
  - returns `GL_TRUE` if `i` has been allocated as a display list

- `glDeleteLists(dls, i)`
  - opposite of `glGenLists()`

- `gllCallLists(n, type, lists)`
  - call several lists, with an offset `i`

  - `gllListBase(i)`
    - used, e.g., for rendering strings when each character would be its own display list

Storing state

- With hierarchical modeling you traverse the model tree recursively in depth-first order

- When you return to a parent node, you want the parent be in the same state it was before the child was processed

- Stacks for storing the state:
  - `glPushAttrib(mask)`
  - `glPopAttrib()`
    - about 20 choices for the mask, `GL_ALL_ATTRIB_BITS` pushes everything (but matrices), and is therefore slower
  - `glPushMatrix()`
  - `glPopMatrix()`
    - works for the matrix of the current matrix mode
Python scripting in Blender

- Blender supports scripting in Python
  - modeling and animation tool (create and modify objects)
  - import and export scripts to other formats than *.blend
  - click documentation
    - Developer documentation for the Python API (a bit old)
    - Blender 2.31 Python API Reference (new)
- Editing
  - edit window SHFT-F11, R-click to open or load a document
  - editing best done with an external editor (e.g., Emacs)
    - after opening a document and editing in Emacs, reload by CTRL-R
- Run by hitting ALT-P in the edit window

Export plugins written in Python

- Take a look at .blender/scripts (in Blender dir)
  - see lot of Python scripts
  - open wrl2export.py
    - it begins
      ```python
      #!BPY
      """ Registration info for Blender menus:
      Name: 'VRML 2.0'
      Blender: 232
      Group: 'Export'
      Submenu: 'All objects...' all
      Submenu: 'Only selected objects...' selected
      Tooltip: 'Export to VRML2 (.wrl) file.'
      """
    ```
  - Then look at File/Export (in Blender)
    - The VRML2.0 exporter is in Python!
    - And you can add your own exporters and they'll show up in the export menu

Example: foot.blend

- A simple scene
  - three boxes, parented to each other
    - use CTRL-P for parenting
  - now if you, e.g. rotate, translate, scale the top-most parent, the children will inherit the transforms
  - notice where the centers of each box are: that's the origin of the local coordinate system (therefore also the rotation axis, etc.)
  - N gives you the numerical data of a selected object
  - when you run the script (ALT-P) the output goes to second Blender window
    - consult the export scripts to see how to write into a file
    - esp. raw_export.py is simple to understand