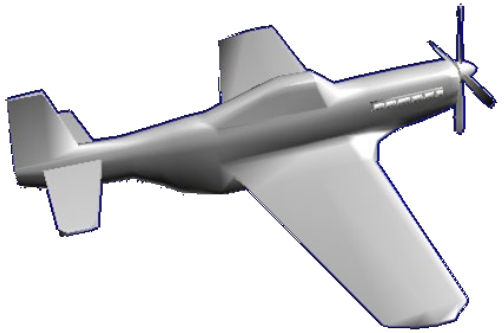
CSc 165
Computer Game Architecture

11 – 3D Modeling for Games

Overview

- Model Characteristics
- 3D Model File Formats
- Model Loaders
- Digital Content Creation (DCC) Tools
- Skinning and UV-unwrapping

Models



Static Data

- 3D geometry (vertex data)
- Polygon (face) data
- Rendering attributes
  - Wireframe / Faceted / Smooth-shaded
  - Lighting & Materials
- Texturing (“skinning”) data

Animation Data (sometimes)

- Model structure (skeletons, joints)
- Model poses
- Animation sequences
  - walk / run / jump / die …

Common 3D Model File Formats

.3ds – 3D Studio Max format

.blend – Blender format

.dae – COLLADA Digital Asset Exchange format

.dem – USGS Standard for Digital Elevation Models

.dxf – Autodesk's AutoCAD format

.hdf – Hierarchical Data Format

.iges – Initial Graphics Exchange Specification

.iv – Open Inventor File Format Info

.lwlo, .lwob & .lwsc – Lightwave 3D file formats

.md2/.md3/.md4/.md5 – Quake Model Files

.ms3d – Milkshape 3D binary format

.msdl – Manchester Scene Description Language

.nff & .enff – (Extended) Neutral File Format

.obj – Alias|Wavefront Object Files

.off – 3D mesh Object File Format

.oogl – Object Oriented Graphics Library

.ply – Stanford Scanning Repository format

.pov – Persistence of Vision ray-tracer

.qd3d – Apple's QuickDraw 3D metafile format

.rkm – RAGE sKeletal Mesh (also used in TAGE)

.viz – used by Division's dVS/dVISE

.vrml – Virtual Reality Modeling Language

.x – Microsoft's DirectX/Direct3D file format

.x3d – eXtensible 3D XML-based scene description format

See wikipedia – list of file formats

.OBJ File Commands

Vertex data

- **`v`** – geometric data
- **`vt`** – texture data
- **`vn`** – vertex normals

Elements

- **`p`** – point
- **`l`** – line
- **`f`** – face
- **`curv`** – curve
- **`surf`** – surface

Grouping

- **`g`** – group name
- **`s`** – smoothing group
- **`mg`** – merging group
- **`o`** – object name

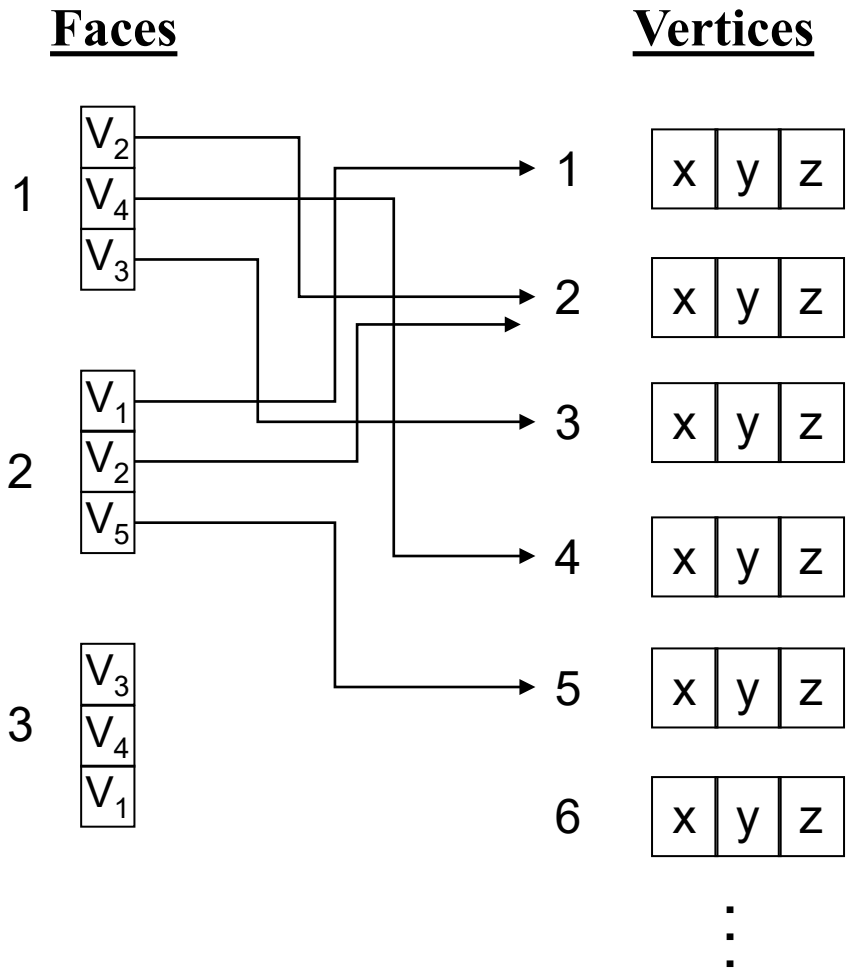
Render Attributes

- **`usemtl`** – material name
- **`mtllib`** – material file name
- **`lod`** – level of detail
- **`shadow_obj`** – shadow casting

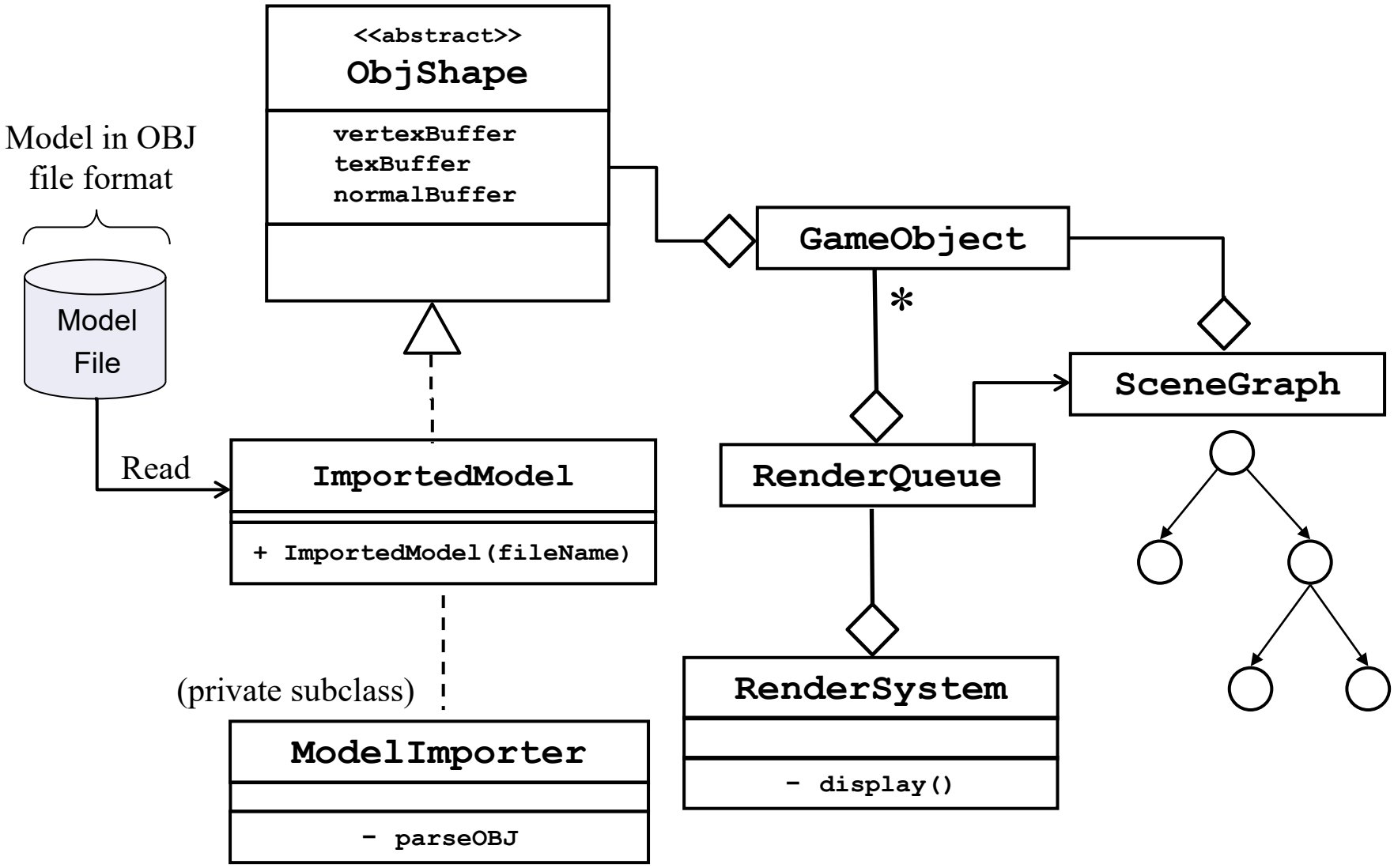
OBJ Example

vertex / texture / normal / face

```
# File 'man.obj'
v  -1.0  -1.0  -1.0
v  -1.0  -1.0   1.0
v  -1.0   1.0  -1.0
...
vt 0.72 0.32
vt 0.86 0.33
...
vn 1.0 0.0 1.0
vn -1.0 0.0 0.5
...
f 2/1/1 4/2/1 3/3/2
f 1/4/2 2/4/2 5/5/3
...
```



Content Loaders in TAGE



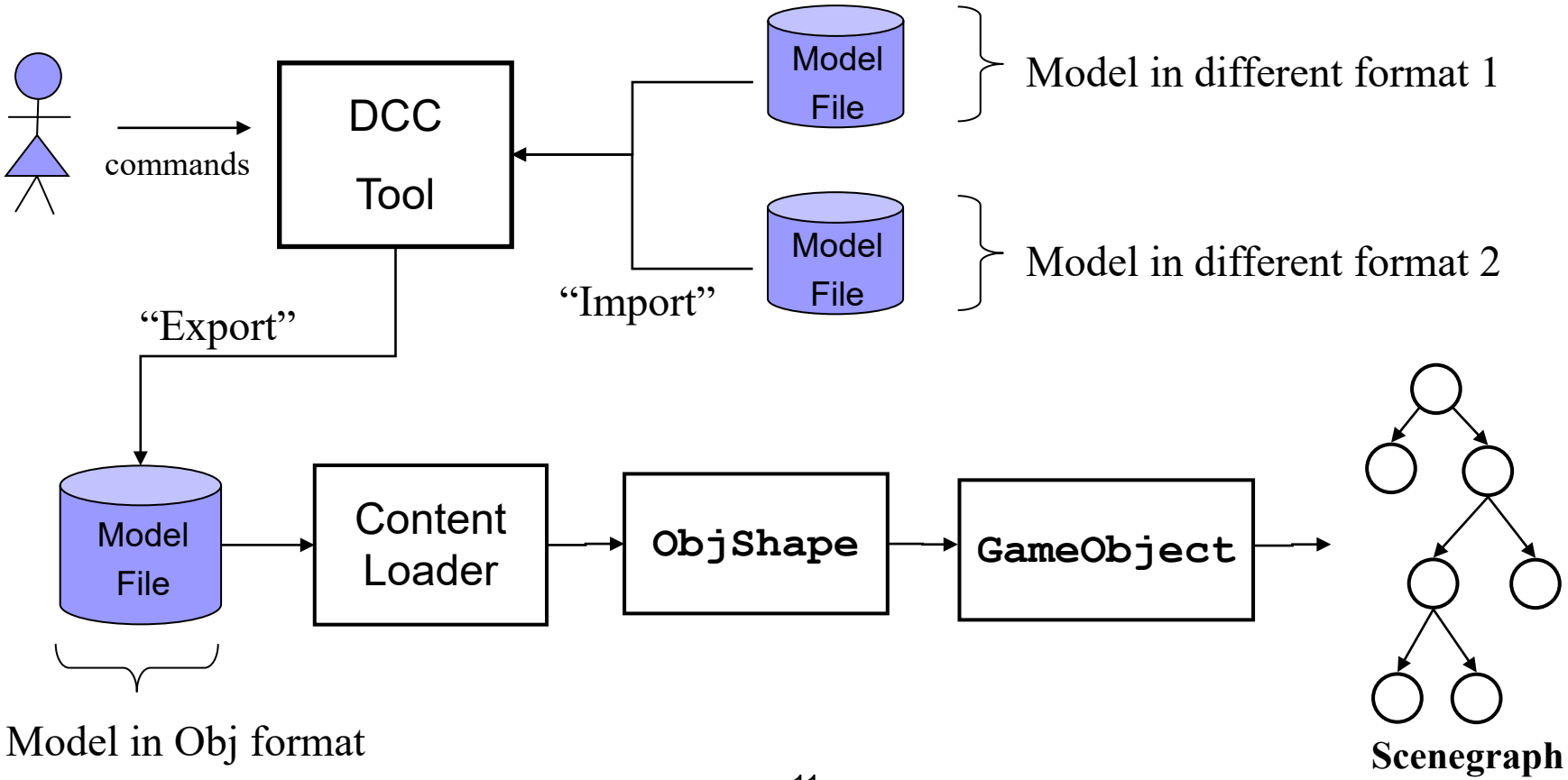
Example: .OBJ Content Loader

```
...
    protected void parseObj(String filename) throw IOException
    {    InputStream input = new FileInputStream(new File(filename));
         BufferedReader br = new BufferedReader(new InputStreamReader(input));
         String line;
         while ((line = br.readLine()) != null)
         {  if(line.startsWith("v "))              // vertex position ("v" case)
            {   for(String s : (line.substring(2)).split(" "))
                {   vertVals.add(Float.valueOf(s));
            }   }
            else if(line.startsWith("vt"))         // texture coordinates ("vt" case)
            ...
            else if(line.startsWith("vn"))         // vertex normals ("vn" case)
            ...
            else if(line.startsWith("f"))          // triangle faces ("f" case)
            {   for(String s : (line.substring(2)).split(" "))
                {   String v = s.split("/")[0];
                    String vt = s.split("/")[1];
                    String vn = s.split("/")[2];

                    int vertRef = (Integer.valueOf(v)-1)*3;
                    int tcRef   = (Integer.valueOf(vt)-1)*2;
                    int normRef = (Integer.valueOf(vn)-1)*3;

                    triangleVerts.add(vertVals.get(vertRef));
                    triangleVerts.add(vertVals.get((vertRef)+1));
                    triangleVerts.add(vertVals.get((vertRef)+2));
                    ...     // same for texture coordinates and normals
         }  }   }
         input.close();
    }...
```

ENGINE

Digital Content Creation (DCC) Tools

Too much work to do "by hand". Use a DCC tool:



DCC Tools

Commercial

- Maya
- Houdini
- Modo
- Lightwave
- etc., etc. …

Free (or nearly)

- Blender
- SketchUp
- Maya / 3DStudio Max for students
- etc., etc., …

Blender

Features:

- Modeling, texturing, lighting, UV-mapping
- Animation rigging/skinning
- Particle, physics, soft-body, fluid & cloth simulations
- Game engine, video post-processing
- Python scripting/plugin support
- Cross-platform (Windows, Linux, Mac OSX, FreeBSD, Solaris….)
- Import/export for a wide variety of model file formats
- Extensive online documentation/wiki support

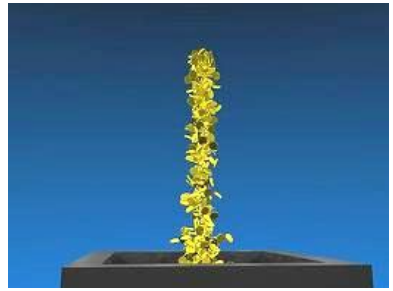
Availability:

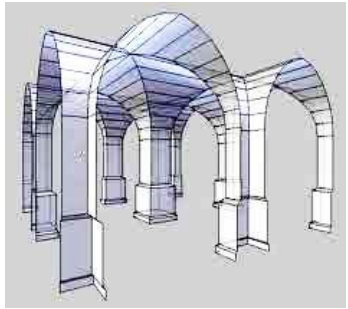
- Free (GNU GPL license), at **`http://www.blender.org`**

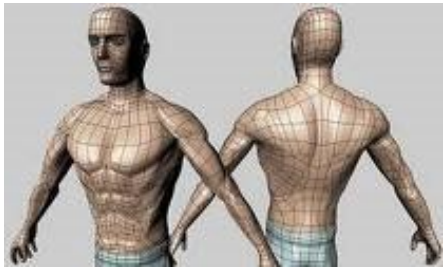
Tutorials:

- **`http://www.blender.org/support/tutorials`**

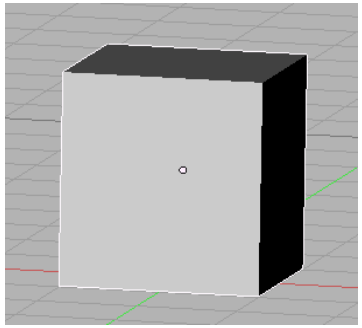
Example Blender Models



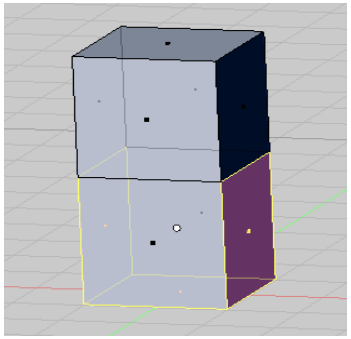




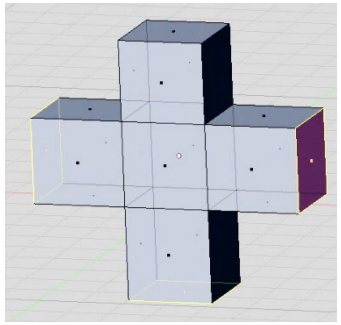
Building Models From Primitives



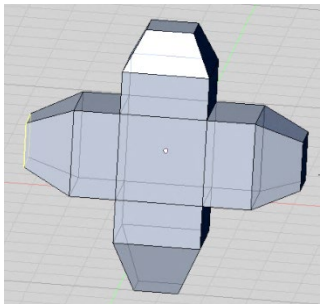
Original Cube



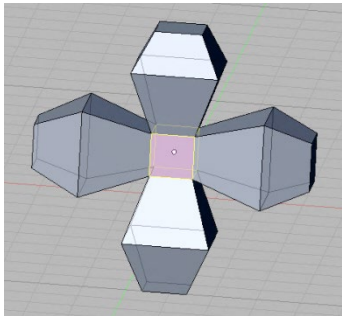
Extrude



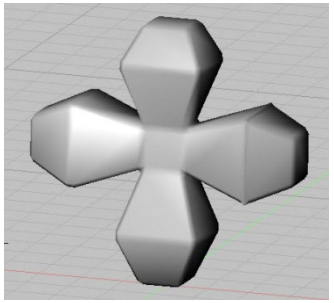
Multiple Extrusions



Scale Ends



Scale Inner Surfaces



Add Bevels and Smooth

Model concept: *The Essential Blender,* Roland Hess, Blender Foundation

Model Exporting

Most DCC tools export a variety of formats

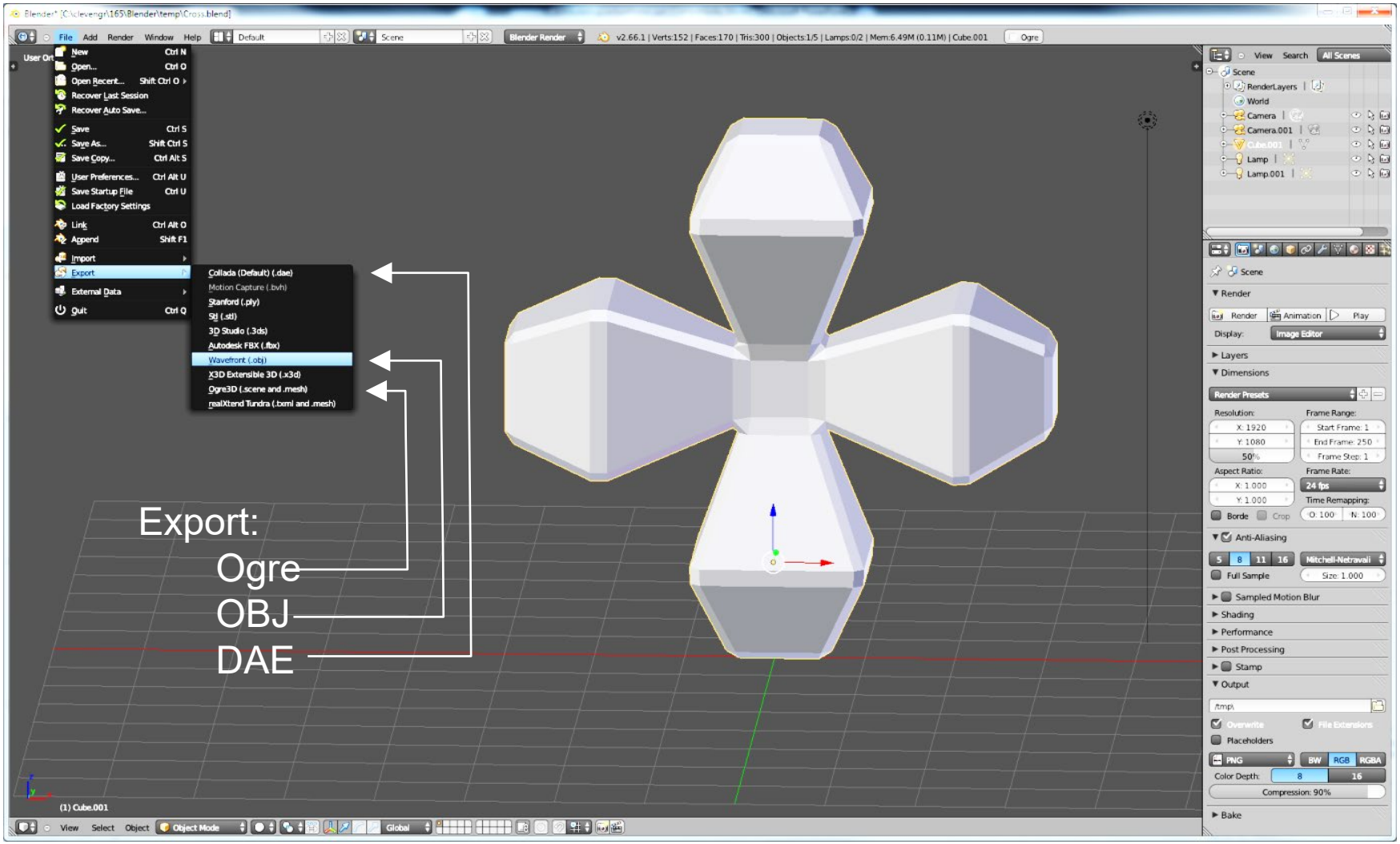
Blender supports exporting to:

- Collada XML (".dae")
- Wavefront OBJ (".obj")
- Stanford Graphics Library (".ply")
- 3DStudio Max (".3ds")
- Lots of others – and more via "add-ons"

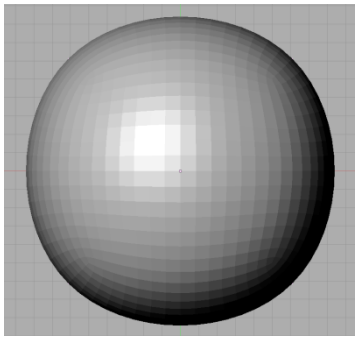
TAGE has ModelLoaders for

- OBJ
- RAGE Skeletal Animation *(Blender exporter add-on)*

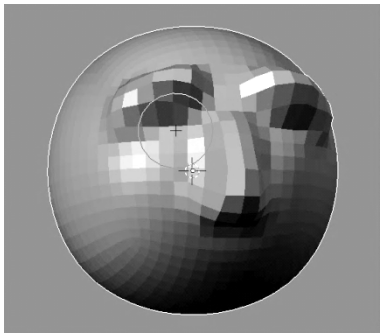
Exporting Models



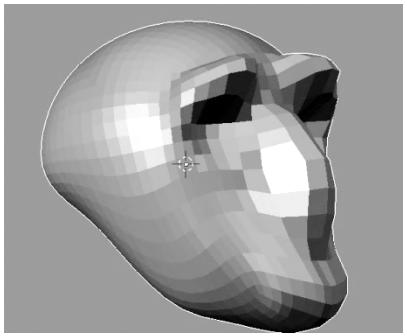
Advanced Modeling: Sculpting



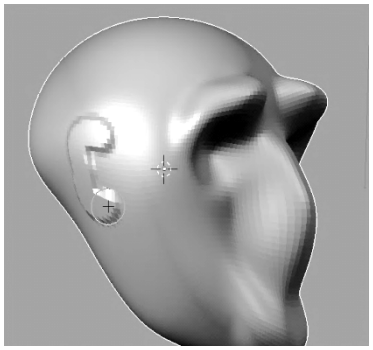
Original sphere



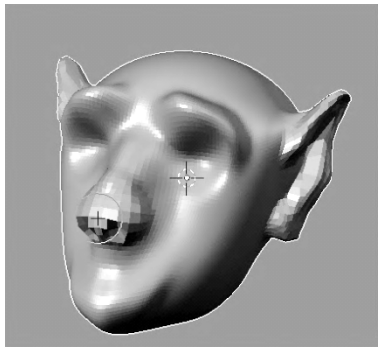
"Add" brush
applied to Sphere



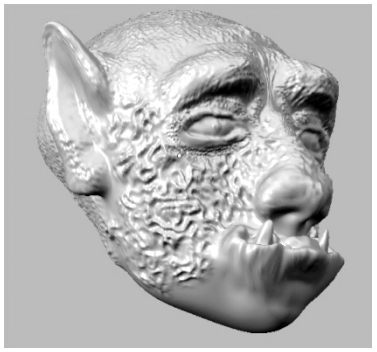
"Grab" brush



"Layer" brush



"Inflate" brush



"Texture" brush

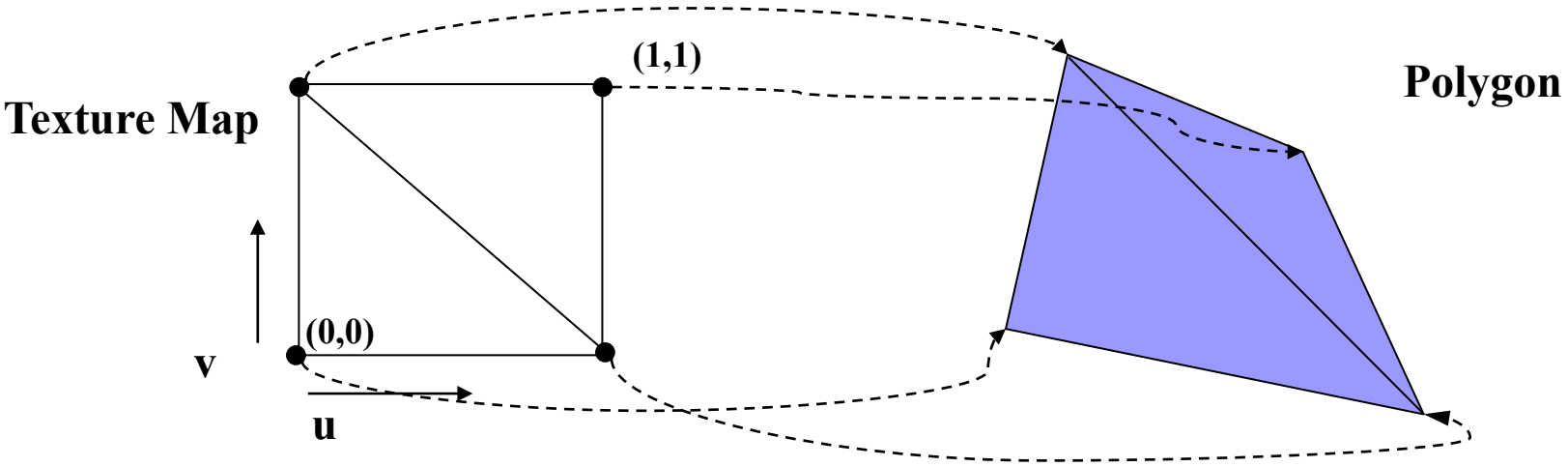
Images from: *The Essential Blender,*
Roland Hess, Blender Foundation

Skinning

Applying “texture” to 3D models

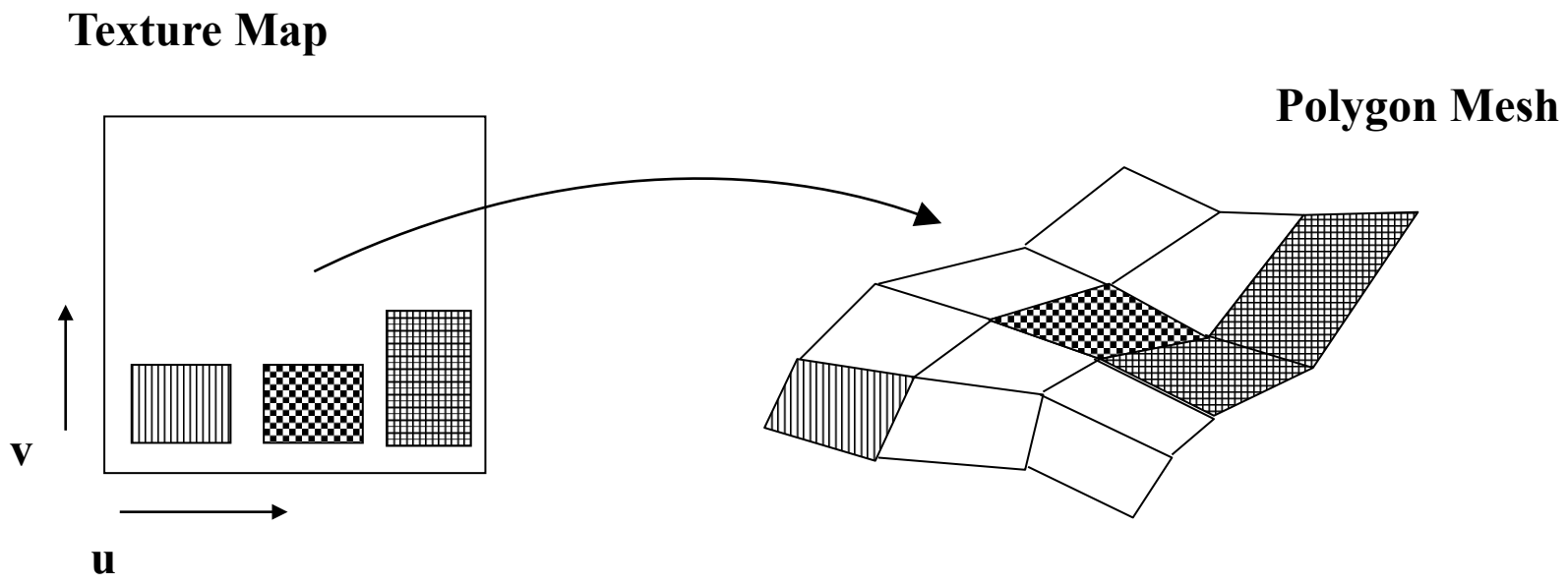
Problem: texture mapping is per-polygon:

Models are *collections* of polygons



Texture Space Subdivision

Texturing *meshes* by *dividing* texture space:



Skinning Example

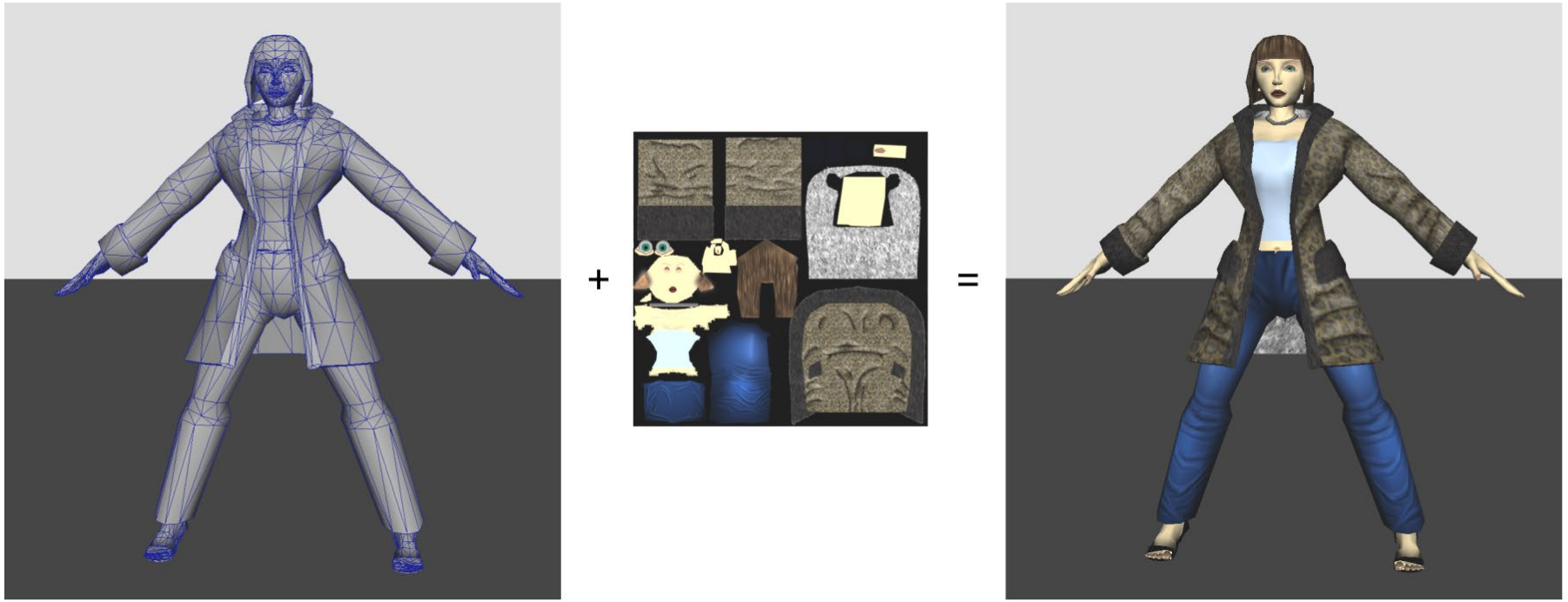


Image credit: Practical Java Game Programming, Clingman et.al., Charles River Media

Texture Subdivision Difficulties

Requires creating a complex mapping
(and the model is often curved)

- How do we create the texture map?
- How do we determine the correct mapping?

Mapping texture locations to model coordinates

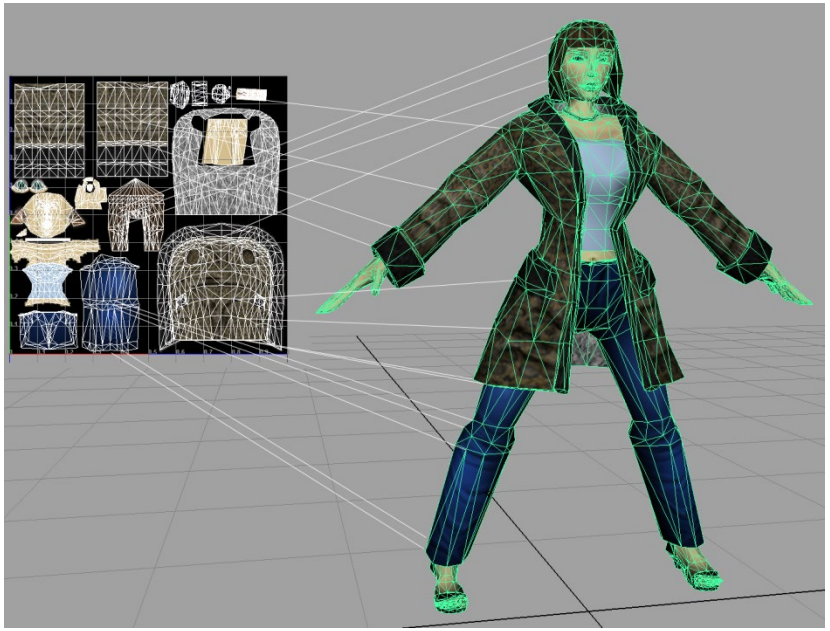


Image credit: Practical Java Game Programming, Clingman et.al., Charles River Media

Creating Texture Skins

- Create the model
- *Flatten* the model ("***UV Unwrapping***")
  - Mark seams
  - Cut along seams ("project")
- Save unwrapped (flattened) UV image
- Save model with texture coordinates
- Use UV image as a "pattern" to create tex map
  - GIMP, PhotoShop, Paint, ...
- Load resulting texture image into game

Example: Soup Can

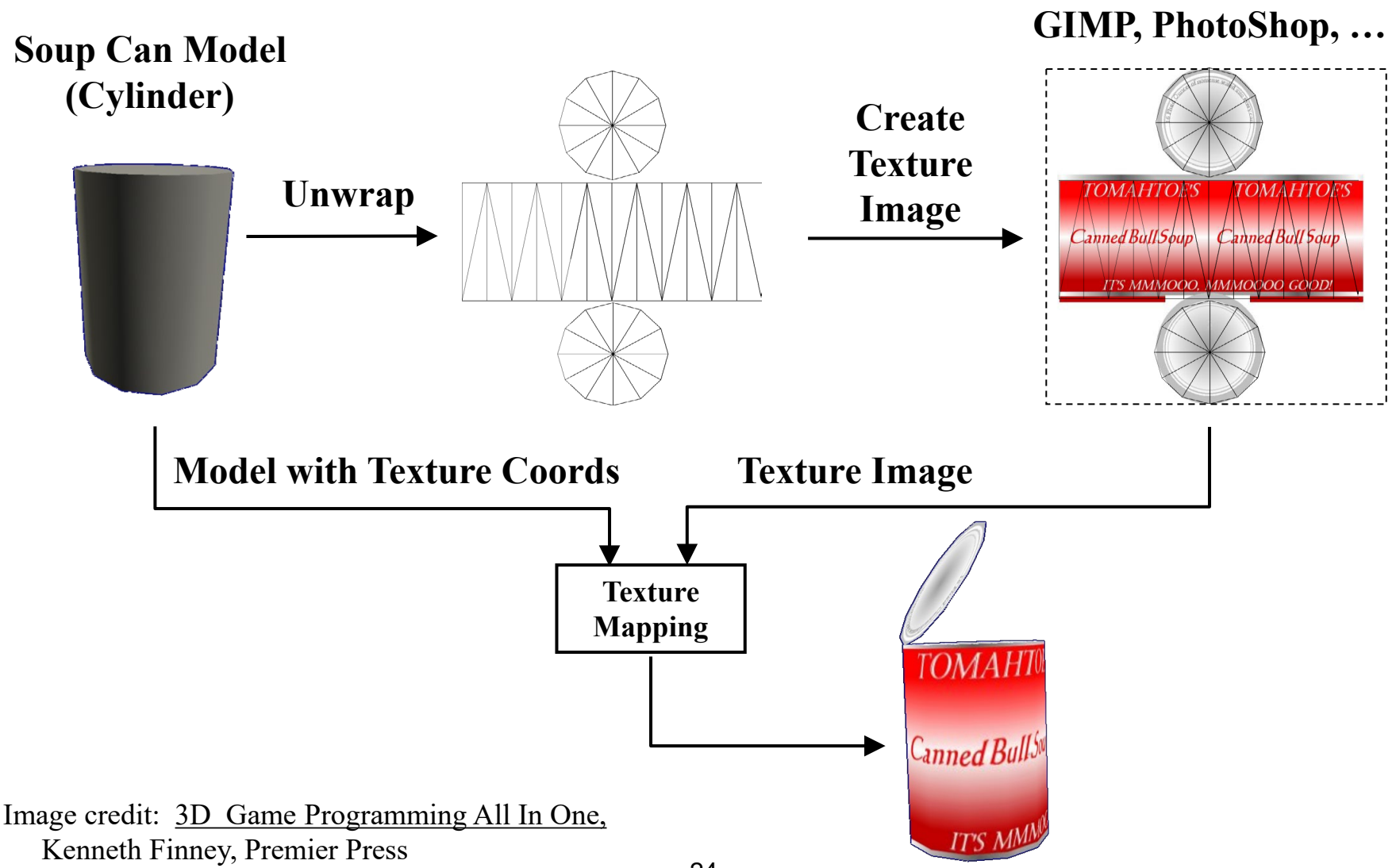
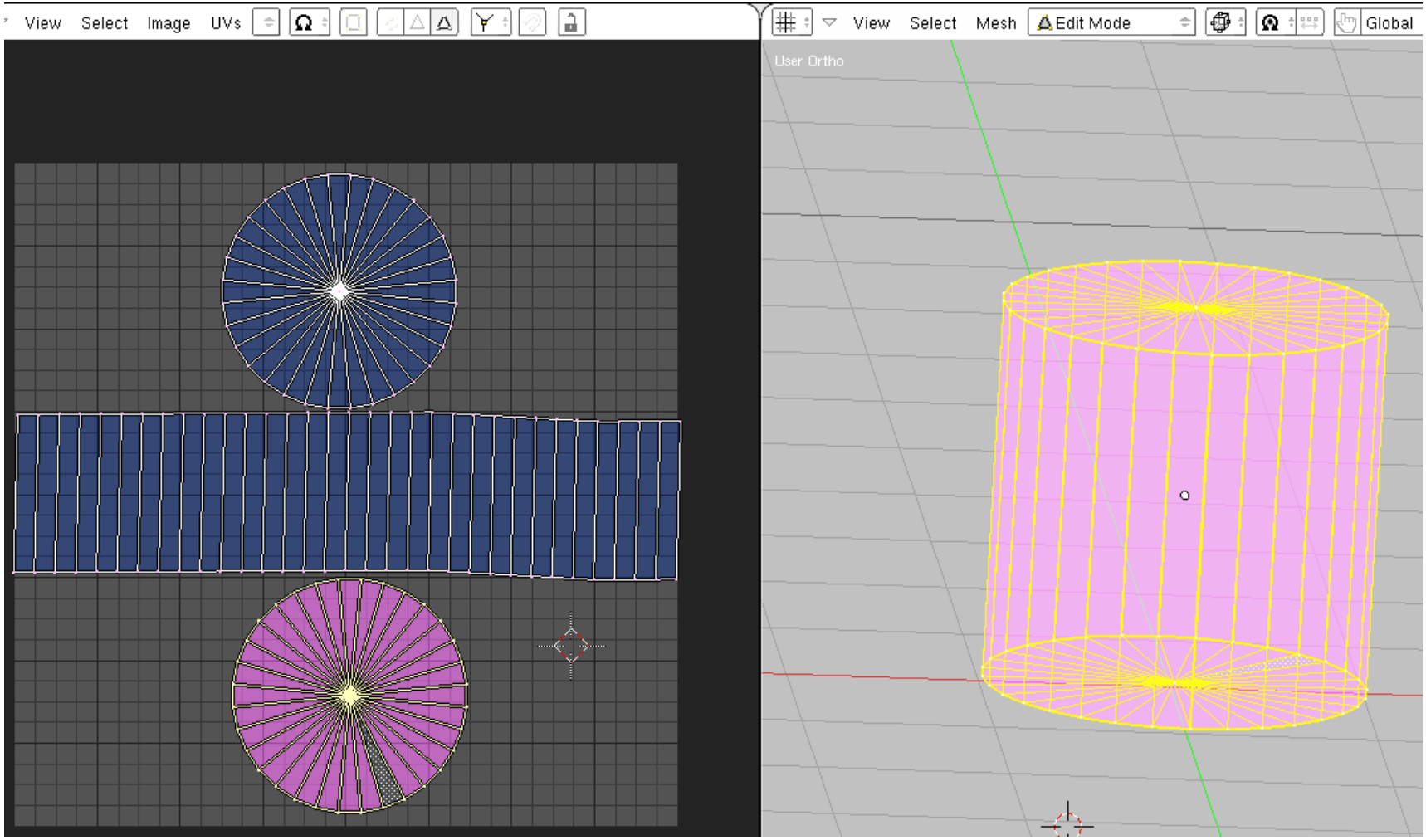


Image credit: 3D Game Programming All In One, Kenneth Finney, Premier Press

Blender UV Unwrapping



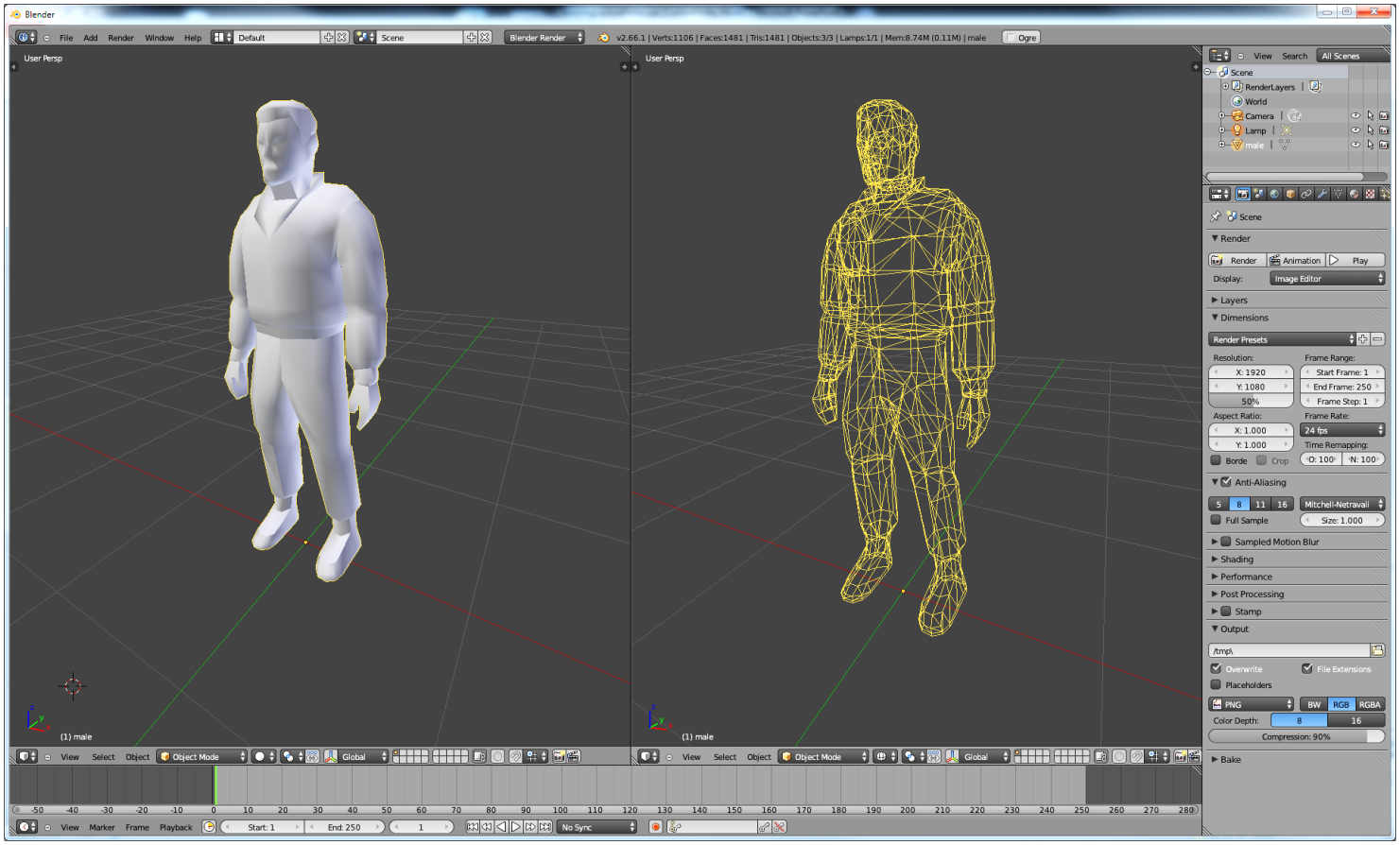
Character Models

More complex, but same approach:

- Create model
- Mark/cut seams (some tools support *groups)*
- Project UV's
- Save projection image
- Use projection image to create texture
- Apply texture to UV map/model

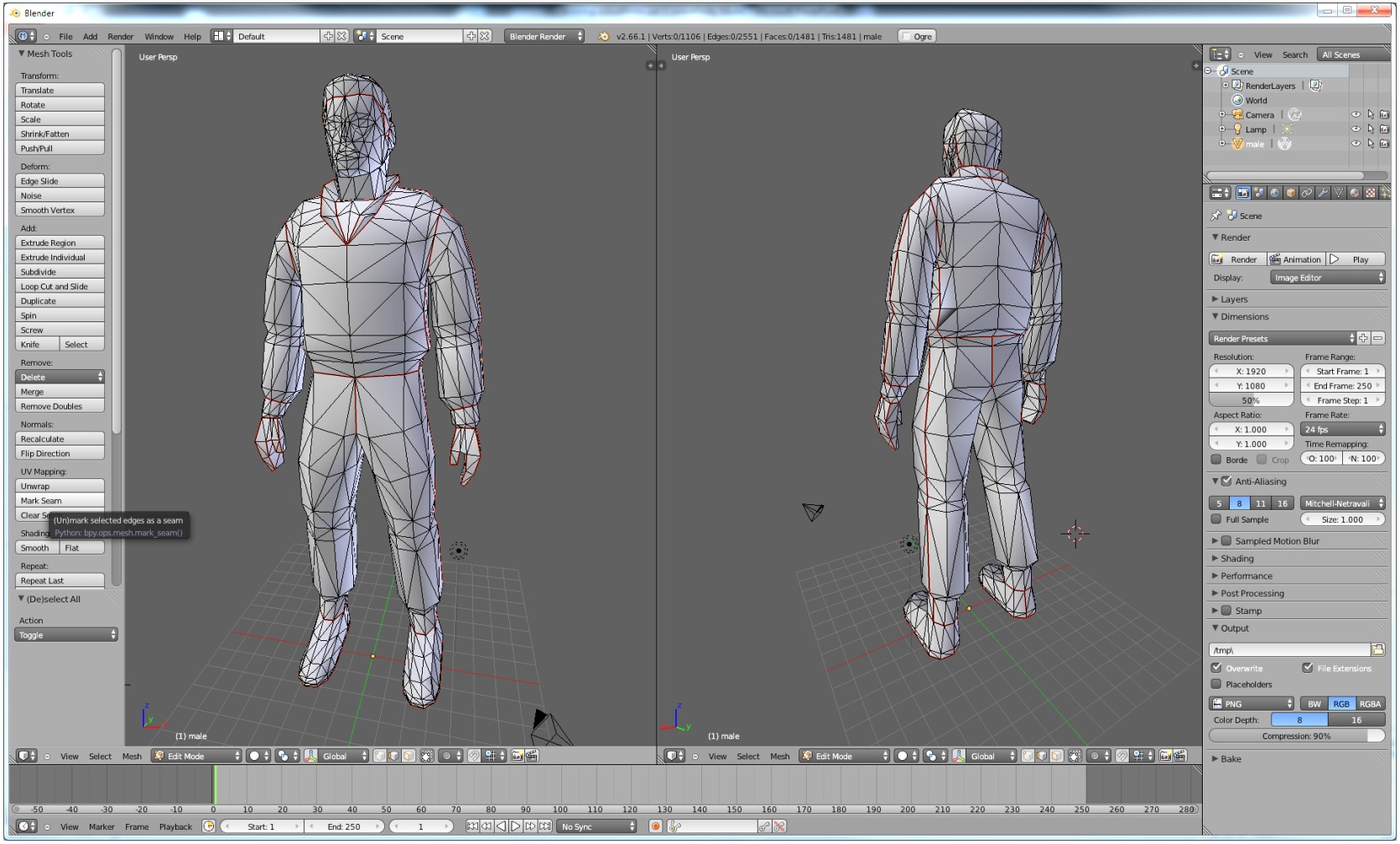
Blender: Character Example

Step 1: Load/Create Model

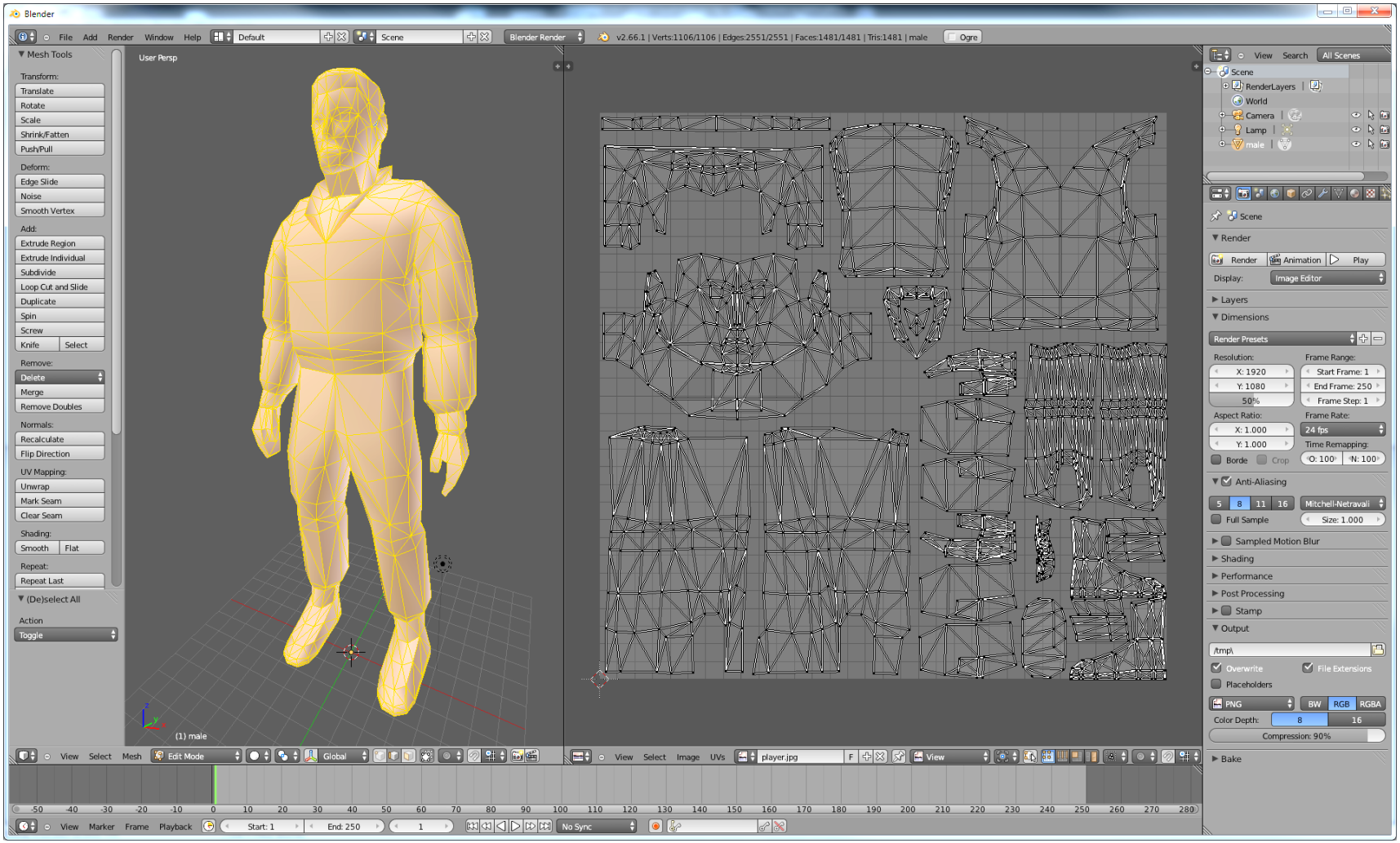


Example from: 3D Game Programming All In One,
Kenneth Finney, Premier Press

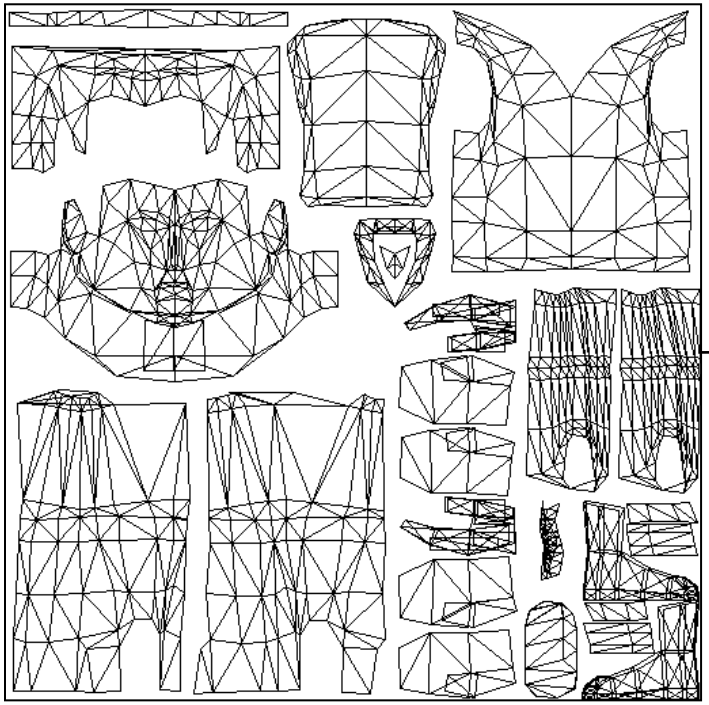
Step 2: Mark Seams



Step 3: Export UVs

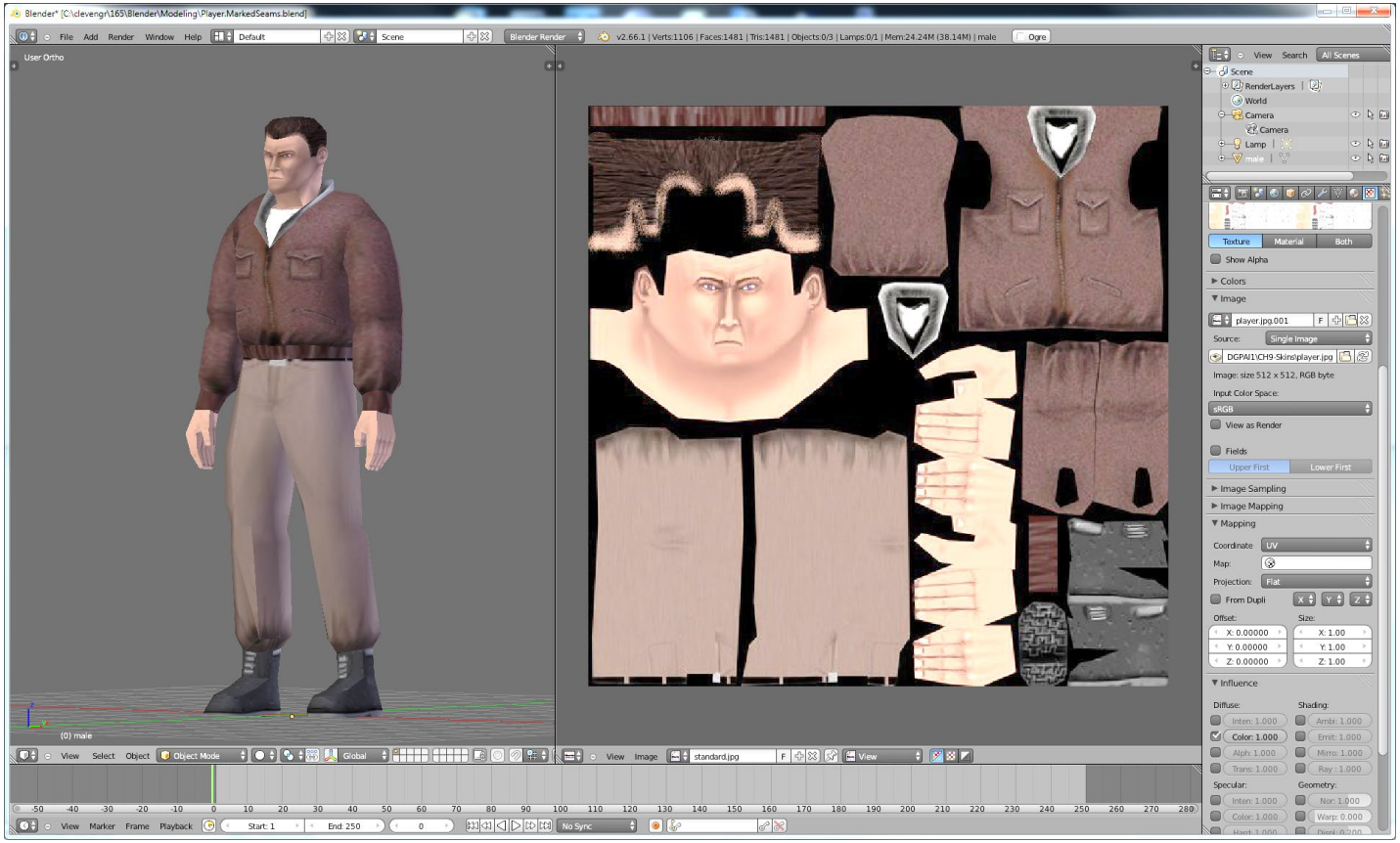


Step 4: Paint Texture



GIMP,
PhotoShop,
MS Paint,
etc.

Step 5: Apply Texture to Model



…in Blender, or directly in TAGE