VripPack User's Guide

Brian Curless
University of Washington

December 9, 2006

These notes are under construction, but should be enough to get you started trying out the vrip package.

1 Installation

Installing and compiling vrip is fairly straightforward. VripPack depends on Tcl/Tk being installed and has been compiled most recently against Tcl/Tk version 8.4. VripPack comes with these libraries for Linux, but you may need still to install your own copy.

If you wish to work under Windows, you can compile with Cygwin. The following web page is a good reference for installing Cygwin and Tcl/Tk as needed for vrip:

```
http://opencircuitdesign.com/cygwin/tcltk.html
```

When installing Cygwin, be sure to include the packages mentioned on that page, as well as much of X11 (probably "safest" to install everything related to X11).
Here are the remaining steps for installing VripPack:

1. Download the VripPack source distribution.
2. Unzip and untar the package. For example:

   ```
   tar xzf vrippack-linux-0.3.tar.gz
   ```

3. If you are not interested in making changes to VripPack and you are running linux, you might be able to run the pre-compiled linux binaries. If so, move on to the next section of this manual.
4. Go into the VripPack directory (e.g., vrippack-0.3) directory and type:
   1. make clobber
   2. make depend
   3. make

   You may get a number of compiler warnings, which you can safely ignore.

2 Environment variables

Before you get started, you will need to set up some environment variables. These environment variables are typically set in the *.cshrc* or *.bash_profile* file in your home directory. We'll use the "setenv" command to describe how to set the environment variables, but bash users will instead use the "export" command.

Note: The Vrip commands are executed as shell scripts using /bin/csh. You *must* set the environment variables in whichever file gets sourced when executing these scripts.

If we call the place where you unpack vrippack the *your_root/vrippack* directory, then you will need to set:

```
setenv VRIP_DIR your_root/vrippack/src/vrip
```

[Note: when unpacking vrippack, it will have a version number as well, e.g., "vrippack-0.3." If you don't change the name to "vrippack," then you'd have to set the root directory described in this section to, e.g., *your_root/vrippack-0.3* .]
In addition, for some versions of tcl and tk, their library scripts must be made known to the software. You can first try to see if the software runs without setting any more environment variables by typing "vrip -noui". If you get no error messages (and the vrip prompt comes up), you can quit vrip (type "quit") and move on.

If, on the other hand, you get en error message complaining about the locations of the Tcl and Tk libraries, then you need to indicate where these library scripts live using the VRIP_TCL_LIBRARY and VRIP_TK_LIBRARY environment variables. If you're using a pre-compiled version of VripPack, then you can set these environment variables as:

```
setenv VRIP_TCL_LIBRARY your_root/vrippack/linux/lib/tcl8.4
setenv VRIP_TK_LIBRARY your_root/vrippack/linux/lib/tk8.4
```

If you compiled VripPack yourself, then you should point the environment variables to the places where your tcl and tk library scripts actually live (e.g., in /usr/lib or /usr/local/lib).

Next, you *may* need to set the library load path to be sure it finds the Tcl and Tk libraries themselves. If you use the pre-compiled binaries or if you compiled using the default library paths, then you might avoid some trouble under linux with:

```
setenv LD_LIBRARY_PATH your_root/vrippack/lib/linux
```

or, under irix:

```
setenv LD_LIBRARY_PATH your_root/vrippack/lib/irix
```

Finally, you need to add the VripPack *bin* directory to your path. For example:

```
set path=($path your_root/vrippack/bin)
```

3 Configuring for your range scanner

Before performing any operations on range data, the *vrip* progam first sources the Tcl code contained in the file *.vriprc* in you home directory. It is here that you will place a few commands to configure the system for your range scanner. These commands will provide parameters to *vrip*; to see what the parameter names

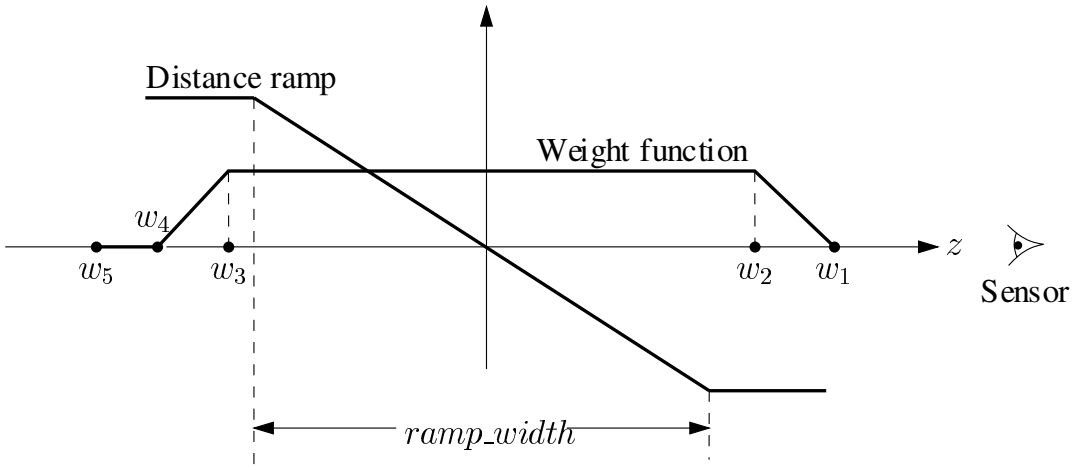

Figure 1: Signed distance and weighting ramps.

and their values, you can type `vrip` followed by the command `vrip_param` (without arguments). We will describe many of these parameters in this section and the next. The remaining parameters are experimental, and we recommend that you do not tamper with their values.

The primary differences among range imaging scanners are the lines of sight of the sensor and the amount of uncertainty and accuracy in the range measurements. Currently, we support two types of viewing frustums: orthographic and cylindrical perspective. An orthographic frustum means that the lines of sight of the sensor are exactly parallel. Cylindrical or line perspective means that the lines of sight are orthographic in one direction and perspective in the others. If your sensor is orthographic (not likely), then you need do no configuration. If your sensor obeys a cylindrical perspective, then we assume that it is perspective in the y-z plane and orthographic in the x-direction. To configure the center of projection in the y-z plane, insert the following line into you *.vriprc*:

```
vrip_param -line_persp_center 0.0 <y-center> <z-center>
```

If your sensor is neither orthographic nor cylindrical perspective, then your best bet is to assume that it is orthographic. Unfortunately, this probably means that you will be unable to do proper space carving and hole filling, as the lines of sight are crucial to these operations. Nonetheless, you should be able to generate reasonable reconstructions of the observered portions of the surface. In all cases, we assyme the "primary" viewing direction is doen the -z axis.

You also need to know the amount of uncertainty in your range data in order to configure the distance and weighting functions as shown in Figure 1. The rule of thumb is that the width of the distance ramp (*ramp_width*) should be about twice the maximum uncertainty in your range measurements.

The weighting function should extend at least the width of the distance ramp, and should feather off on either end. Letting the weight persist a bit in front of the surface permits some extra carving and robustness against local distortions and outliers. The weight function should drop rapidly behind the surface to avoid interference from opposing surfaces.

For example, if we had an uncertainty interval of about 1 mm, we might configure our distance and weight functions as:

```
vrip_param -ramp_width 0.002
vrip_param -w1 0.003 -w2 0.0025 -w3 -0.001 -w4 -0.0015 -w5 -0.002
```

The last weight (w_5) exists primarily for historic reasons and has little impact. In a future release, this weight will vanish; for now, make sure that it is slightly larger (in absolute value) than w_4.

4 Range images, range surfaces, and weights

Vrip reads range data stored as "ply" files and has the ability to create range surfaces from range images and assign weights to the vertices.

4.1 Ply files

Software for reading and writing your own ply files is available for download[1]. This software has some documentation to get you started, but it does not discuss the particulars of how range images are stored. If you are unfamiliar with the ply format, then you should study its documentation before reading the remainder of this section.

A range image stored in a ply file consists of two "element" types: *vertex* and *range_grid*. The *vertex* elements have three "properties" - *x*, *y*, *z* - corresponding to the vertex positions. The vertex elements are stored sequentially in an order such that the *range_grid* can index into them.

The *range_grid* is a matrix that indicates what point(s) if any were seen along each line of sight of the regular sampling grid. *Range_grid* elements have one property - a list of vertices per line of sight in the range image. Normally, this list is either of length 0, meaning no range point was observed or length 1 indicating a single point was observed. If the length is 1, then the list contains the index of the vertex observed. The *range_grid* elements are stored in row major order.

In order to indicate the dimensions of the range image, you need to include the strings "num_cols #" and "num_rows #" in the "obj_info" field of the ply file. Your setup structures for writing the range grid will look something like this:

```
struct PlyVertex {
  float x,y,z;
};

struct RangeGridPntt {
  unsigned char num_pts;
  int *pts;
};

PlyProperty vert_props[] = {
  {"x", PLY_FLOAT, PLY_FLOAT, offsetof(PlyVertex,x), 0, 0, 0, 0},
  {"y", PLY_FLOAT, PLY_FLOAT, offsetof(PlyVertex,y), 0, 0, 0, 0},
  {"z", PLY_FLOAT, PLY_FLOAT, offsetof(PlyVertex,z), 0, 0, 0, 0}
}
```

[1] file://www-graphics.stanford.edu/pub/zippack/ply-1.1.tar.Z

```
PlyProperty range_props[] = {
  {"vertex_indices", PLY_INT, PLY_INT, offsetof(RangeGridPnt,pts),
   1, PLY_UCHAR, PLY_UCHAR, offsetof(RangeGridPnt,num_pts)},
};
```

The header of ply files is in ascii, so you can always check whether or not you're writing out the header as desired. Your range grid should have a header like:

```
ply
format binary_big_endian 1.0
obj_info num_cols 512
obj_info num_rows 400
element vertex 22310
property float x
property float y
property float z
element range_grid 204800
property list uchar int vertex_indices
end_header
```

In addition, you can set the file type to ascii before writing and then inspect the data. To see some example data, you can look at one of the smaller datasets, such as the drill bit in the Stanford 3D Scanning Repository[2].

4.2 Converting range images to range surfaces

Before merging range data, *vrip* first converts each range image into a range surface; i.e., it creates a triangular tesselation by connecting nearest neighbors in the range image. Some neighbors, however, are far apart, indicating a depth discontinuity. No triangles should be created over depth discontinuities, because they are likely to be erroneous.

You have a choice of two methods for avoiding these bad tesselations. Neighboring vertices in the range image are connected if the resulting triangle meets one of two criteria: (1) the dot product between the triangle normal and the viewing direction is below a threshold value, or (2) the lengths of all edges of the triangle are below a threshold value. The default is to use the orientation criterion, but you can choose instead to use the edge length criterion with:

```
vrip_param -use_length 1
```

When using the orientation criterion, you can set the threshold (default is 0.15 or about 81 degrees) with the command:

```
vrip_param -min_view_dot <float>
```

The view direction is taken to be the vector, $\mathbf{v}$ = (0,0,-1), and the dot product is negated before comparing to the threshold. Note that this usage of a fixed view direction is only stricly correct when the sensor is orthographic and looking down the -z axis. The optimal approach would be to consult the actual directions of the viewing rays in the vicinity of each triangle. This approach has yet to be implemented, but the orthographic approximation will give reasonable results if the viewing rays of your scanner do not diverge severely.

When using the edge length criterion, you can set the threshold (default is 0.003) with the command:

[2] http://www-graphics.stanford.edu/data/3Dscanrep

```
vrip_param -max_edge_length <float>
```

Note that you do not have to use the range image tesselator inside of *vrip*; *vrip* can also merge pre-tesselated range surfaces, as long as they are in the ply format and the vertex indices are ordered counter-clockwise. However, as part of the reconstruction process, *vrip* resamples the range surface before merging it into the volume. The resampling effectively yields another range image for which the tesselation criterion still applies. Therfore, even if you are providing pre-tessellated range surfaces as input, *you must select a suitable tesellation criterion*.

4.3 Assigning vertex weights

Once a range surface has been tesselated, *vrip* will proceed to assign weights to the vertices. The formula for vertex weight is:

$$W = W_v \cdot W_b$$

where W_v is the view weight and W_b is the boundary weight. The view weight is defined by:

$$W_v = (\mathbf{v} \cdot \mathbf{n})^{\alpha_v}$$

where $\mathbf{v}$ is the view direction, $\mathbf{n}$ is the normal at the vertex, and α_v controls how rapidly the view weight falls off with obliquity to the sensor. You can modify the value of α_v (default is 2) with:

```
vrip_param -view_weight_exp <float>
```

The boundary weight is based on the number of steps (edge traversals) a vertex is from the boundary of the mesh. The equation for boundary weight is:

$$W_b = \begin{cases} 1 & \text{if } S_b \geq S_{max} \\ \left(\frac{S_b}{S_{max}}\right)^{\alpha_b} & \text{if } S_b < S_{max} \end{cases}$$

where S_b is the number of edge steps a vertex is from the boundary, S_{max} is a threshold that determines how far the boundary downweighting should diffuse into the mesh, and α_b controls how rapidly the boundary weight falls off with proximity to the boundary. You can modify the value of α_b (default is 1) with:

```
vrip_param -boundary_weight_exp <float>
```

and you can modify the value of S_{max} (default is 8) with:

```
vrip_param -max_boundary_steps <int>
```

Note that if you use a pre-tessellated range surface as an input, then *vrip* will assign the weights for you. You also have the option of assigning vertex weights beforehand. To pass the weights to *vrip*, you need to store them in the "confidence" field with each vertex. (The name of this field is "confidence" rather than "weight" for historical reasons.)

4.4 Previewing the tessellated, weighted range surface

If you want to see how the tessellation and weight assignment turned out, you can simply start *vrip* (by typing `vrip`), and then execute the command:

```
vrip_plyconf <in.ply> <out.ply>
```

The input file will be tessellated to make a range surface (if necessary), weights will be assigned to the vertices, and the resulting mesh will be written to the output file.

5 Merging range images

We assume that all of your range surfaces are already aligned with respect to each other. This alignment information can be provided in one of two ways. The simplest is to create a file called MANIFEST in each directory where the range data lives. This file should contain a list of all the .ply files to merge. In addition, each .ply should have a .xf file associated with it (e.g., scan1.ply and scan1.xf). Each .xf should be an ASCII file containing the 4x4 matrix that describes the rotation and translation alignment for a given .ply file.

Alternatively, you can provide a single file (we call it a *configuration* or *conf* file) which contains the list of meshes and their transformations. The transformations should be of the form of 7 floating point numbers; the first 3 are the translation and the last 4 are a quaternion representing the rotation:

```
bmesh <range_image> <tx> <ty> <tz> <qi> <qj> <qk> <ql>
```

These transformations should have the following property: if you were to rotate each mesh by the quaternion $(q_i,\ q_j,\ q_k,\ q_l)$ and then translate it by $(t_x,\ t_y,\ t_z)$, then the meshes would all be in the same coordinate system.

Now, to construct the volume and begin merging range images you type:

```
vripnew <name.vri> <conf_file> <bound_volume> <res_in_meters>
```

The `<conf_file>` will either be `MANIFEST` or the name of the conf file in which you enumerated the meshes and the transformations.

Vrip must allocate a (sparse) volumetric grid before merging meshes. It uses the `<bound_volume>` argument for this purpose. The `<bound_volume>` may be a ply file that is the same size as or larger than the object you wish to reconstruct. Alternatively, you can repeat `<conf_file>` instead, and vrip will sift through all your ply files to determine a bounding volume large enough to encompass them when they are transformed into one coordinate system.

Executing vripnew will cause a new volumetric grid to be created with voxel size `res_in_meters` such that it surrounds the bounding mesh you provided. It will also begin merging the range images specified in the conf file.

To add more range images, simply type:

```
vripupdate <name.vri> <conf_file>
```

and the volumetric grid will be updated in place.

There are a few options you can provide to these commands. To find out what they are, type the command name without arguments. The carve options will empty out the space around the range surfaces. We will have a more in depth discussion of the range carving issues at a later time (under construction).

To view the volume, type:

```
vripslicer <name.vri>
```

[Note: you must have an X display set for this to work.]

The top row of radio buttons shows in which direction you are taking slices. The slice slider shows which slice you are looking at. The color coding starts up as black = "empty", gray = "near the surface", and brown = "unseen" or "not touched". You can also select "show weights". This turns the color scheme into two main channels: blue = weight and red/green = distance.

5.1 Resolution vs. ramp width

The voxel grid resolution (`res_in_meters`) has a tricky interplay with the parameters for the distance and weight ramps. The distance ramp is ideally chosen to be as narrow as twice the maximum uncertainty in the scanner. The resolution then has to be chosen small enough that it adequately samples this ramp, even with significant variations in surface orientation. If the resolution is not chosen to be small enough, then unwanted holes may appear in the reconstruction. The upshot is that you can reduce the number of holes by either increasing the resolution (by using a smaller number for `res_in_meters`) or by widening the ramps. The trade-off is as follows: smaller voxel size yields more triangles, while widening the ramps leads to more interference between opposing surfaces.

6 Extracting a surface

To extract a surface, simply type:

```
vripsurf <name.vri> <name.ply>
```

This will run the marching cubes algorithm and give you a surface in the form of a ply file. To remove small triangles that yield rendering artifacts, add the `-remove_slivers` option. This option is still in the experimental stages; if it gives you problems, then leave it out and try another conservative decimator. Typical mesh reductions are around 2:1. The remaining options relate to hole filling, which will be described in detail at a later date.