Urban Chaos File Formats

PieroZ

May 23, 2024

Contents

1 File Formats

1.1 .vue

1.1.1 Description

The .vue file format is used in Urban Chaos to store transformation data for different body parts in a given frame. This format is used in the keyframe editor, as it accepts it as an input file and allows user to modify existing and create new animations based from keyframes defined in the .vue.

1.1.2 Example

```
frame 0
transform "PELVIS00" 1.000 0.000 0.000 0.000 1.000 0.000 0.000 0.000 1.000 -0.032 52.764 41.197
transform "Lfemur00" 1.000 0.000 0.000 0.000 1.000 0.000 0.000 0.000 1.000 3.320 55.007 36.521
transform "Ltibia00" 1.000 0.000 0.000 0.000 1.000 0.000 0.000 0.000 1.000 3.342 57.468 21.061
transform "Lfoot00" 1.000 0.000 0.000 0.000 1.000 0.000 0.000 0.000 1.000 3.194 59.038 4.386
transform "Rfemur00" 1.000 0.000 0.000 0.000 1.000 0.000 0.000 0.000 1.000 -3.133 54.651 36.498
transform "Rtibia00" 1.000 0.000 0.000 0.000 1.000 0.000 0.000 0.000 1.000 -2.994 57.717 20.980
transform "Rfoot00" 1.000 0.000 0.000 0.000 1.000 0.000 0.000 0.000 1.000 -2.871 59.190 4.258
transform "torso00" 1.000 0.000 0.000 0.000 1.000 0.000 0.000 0.000 1.000 -0.085 54.542 42.202
transform "Rhumorus00" 1.000 0.000 0.000 0.000 1.000 0.000 0.000 0.000 1.000 -7.195 55.968 54.408
transform "Rradius00" 1.000 0.000 0.000 0.000 1.000 0.000 0.000 0.000 1.000 -8.953 57.263 45.458
transform "Rhand00" 1.000 0.000 0.000 0.000 1.000 0.000 0.000 0.000 1.000 -8.585 54.128 35.677
transform "Lhumorus00" 1.000 0.000 0.000 0.000 1.000 0.000 0.000 0.000 1.000 6.965 55.808 54.364
transform "Lradius00" 1.000 0.000 0.000 0.000 1.000 0.000 0.000 0.000 1.000 8.675 57.130 45.445
transform "Lhand00" 1.000 0.000 0.000 0.000 1.000 0.000 0.000 0.000 1.000 8.519 53.630 35.678
transform "skull00" 1.000 0.000 0.000 0.000 1.000 0.000 0.000 0.000 1.000 0.042 53.877 58.805
```

Listing 1: frame 0 from roper.vue

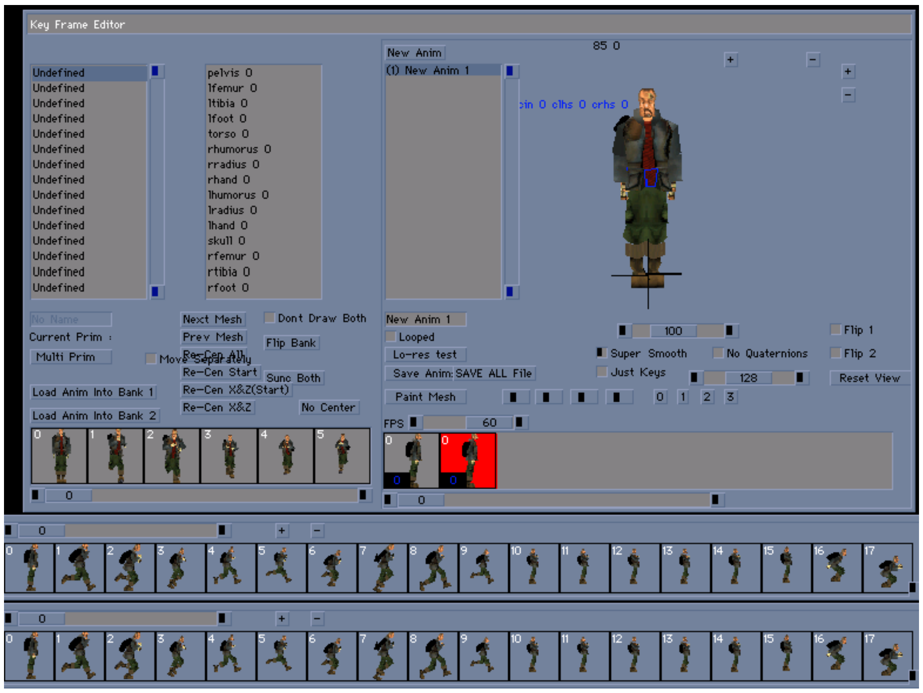


Figure 1: roper.vue Frame 0 previewed in Key Frame Editor

```
frame 1
transform "PELVIS00" 0.986 0.149 0.078 -0.156 0.984 0.087 -0.064 -0.098 0.993 0.002 52.764 41.211
transform "Lfemur00" 0.995 0.019 0.101 0.005 0.973 -0.230 -0.103 0.230 0.968 3.256 55.929 37.025
transform "Ltibia00" 0.995 0.018 0.101 -0.103 0.196 0.975 -0.002 -0.980 0.197 4.878 54.772 21.497
transform "Lfoot00" 1.000 0.006 -0.004 0.007 -0.477 0.879 0.003 -0.879 -0.477 4.605 71.427 19.737
transform "Rfemur00" 0.999 -0.019 -0.033 -0.005 0.787 -0.616 0.038 0.616 0.787 -3.048 54.619 36.467
transform "Rtibia00" 0.999 -0.024 -0.035 0.007 0.909 -0.416 0.041 0.415 0.909 -3.516 47.470 22.363
transform "Rfoot00" 0.990 -0.139 0.027 0.132 0.833 -0.537 0.052 0.535 0.843 -4.072 41.862 6.550
transform "torso00" 0.942 -0.335 0.016 0.325 0.923 0.205 -0.084 -0.188 0.979 -0.391 54.407 42.360
transform "Rhumorus00" 0.996 -0.042 -0.073 0.058 0.974 0.221 0.062 -0.224 0.973 -7.648 55.812 54.482
transform "Rradius00" 0.959 0.006 -0.284 -0.269 0.338 -0.902 0.091 0.941 0.325 -9.881 59.154 46.193
transform "Rhand00" 0.953 0.174 -0.248 -0.261 0.056 -0.964 -0.153 0.983 0.098 -9.571 48.892 45.732
transform "Lhumorus00" 0.449 -0.810 0.377 0.887 0.352 -0.301 0.111 0.469 0.876 5.642 50.924 54.635
transform "Lradius00" 0.401 -0.910 0.109 -0.260 -0.227 -0.939 0.879 0.348 -0.327 6.592 45.819 47.069
transform "Lhand00" 0.053 -0.949 0.311 -0.538 -0.289 -0.791 0.841 -0.126 -0.526 -1.144 43.360 53.533
transform "skull00" 0.999 -0.001 0.046 -0.005 0.990 0.138 -0.046 -0.138 0.989 -1.879 50.627 58.472
```

Listing 2: frame 1 from roper.vue

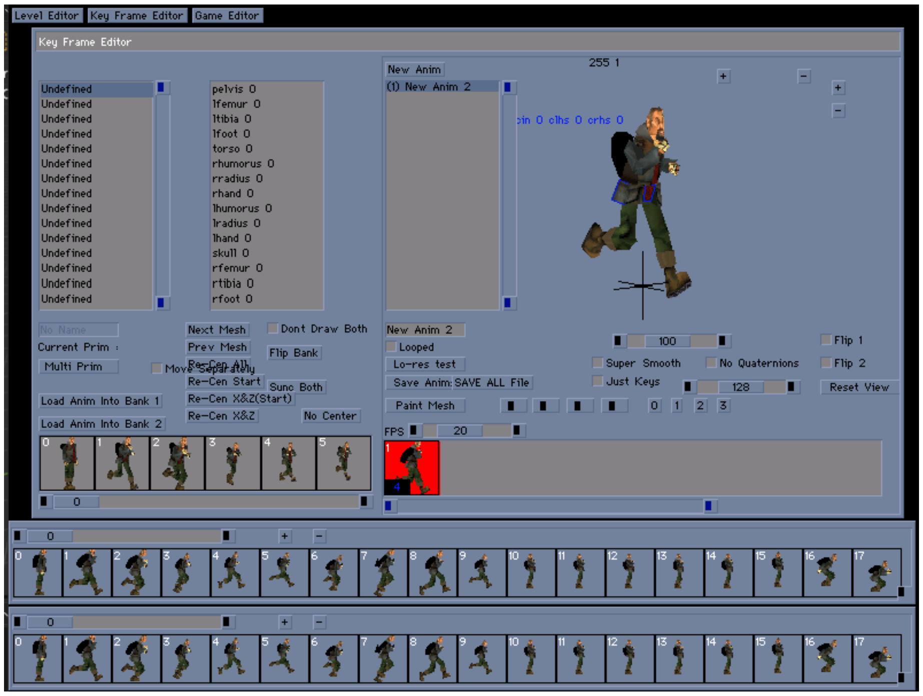


Figure 2: roper.vue Frame 1 previewed in Key Frame Editor

1.1.3 Additional Information

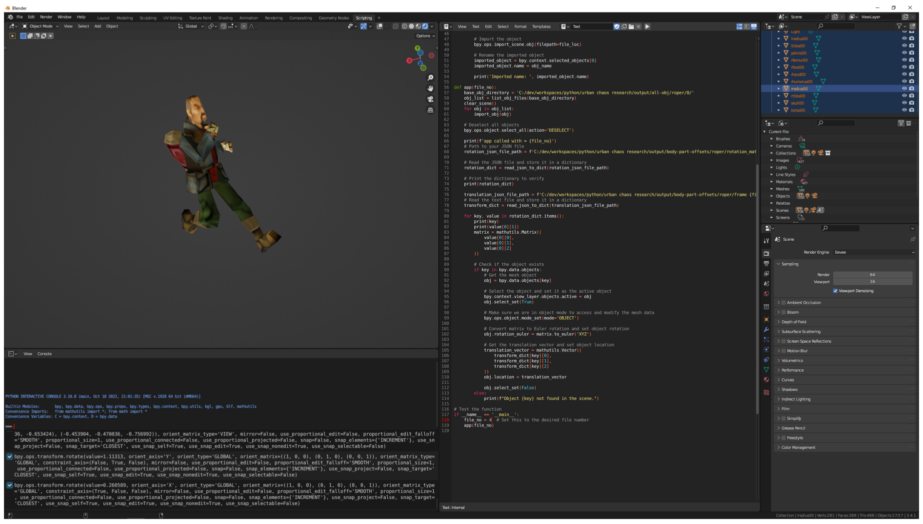
The .vue files are critical in defining the positioning and orientation of various body parts at specific frames in the animation sequence. Each transform entry defines the transformation matrix and the translation vector for a body part, allowing for precise control over the animations. Unlike .all file .vue does not contain mesh data. While loading .vue file into the key frame editor the mesh data is acquired from .sex file. The .sex filename must correspond to .vue filename in order for editor to load it.

1.1.4 Math

The math behind .vue is as follows: First 9 values create a rotation matrix. Let's take pelvis00 from frame 1 as an example:

$$R = \begin{bmatrix} 0.986 & 0.149 & 0.078 \\ -0.156 & 0.984 & 0.087 \\ -0.064 & -0.098 & 0.993 \end{bmatrix}$$

Similar values can be acquired extracting data directly from roper.all file. However it seems that frame ids do not correspond to each other between .vue and .all files. Below image has been obtained by applying transformations from 5th element from roper.all file.



```python
        # Import the object
        bpy.ops.import_scene.obj(filepath=file_loc)

        # Rename the imported object
        imported_object = bpy.context.selected_objects[0]
        imported_object.name = obj_name

        print('Imported name: ', imported_object.name)

def app(file_no):
    base_obj_directory = 'C:/dev/workspaces/python/urban chaos research/output/all-obj/roper/0/'
    obj_list = list_obj_files(base_obj_directory)
    clear_scene()
    for obj in obj_list:
        import_obj(obj)

    # Deselect all objects
    bpy.ops.object.select_all(action='DESELECT')

    print(f'app called with = {file_no}')
    # Path to your JSON file
    rotation_json_file_path = f'C:/dev/workspaces/python/urban chaos research/output/body-part-offsets/roper/rotation_mat

    # Read the JSON file and store it in a dictionary
    rotation_dict = read_json_to_dict(rotation_json_file_path)

    # Print the dictionary to verify
    print(rotation_dict)

    translation_json_file_path = f'C:/dev/workspaces/python/urban chaos research/output/body-part-offsets/roper/frame {fi
    # Read the text file and store it in a dictionary
    transform_dict = read_json_to_dict(translation_json_file_path)

    for key, value in rotation_dict.items():
        print(key)
        print(value[0][1])
        matrix = mathutils.Matrix((
            value[0][0],
            value[0][1],
            value[0][2]
        ))

        # Check if the object exists
        if key in bpy.data.objects:
            # Get the mesh object
            obj = bpy.data.objects[key]

            # Select the object and set it as the active object
            bpy.context.view_layer.objects.active = obj
            obj.select_set(True)

            # Make sure we are in object mode to access and modify the mesh data
            bpy.ops.object.mode_set(mode='OBJECT')

            # Convert matrix to Euler rotation and set object rotation
            obj.rotation_euler = matrix.to_euler('XYZ')

            # Get the translation vector and set object location
            translation_vector = mathutils.Vector((
                transform_dict[key][0],
                transform_dict[key][1],
                transform_dict[key][2]
            ))
            obj.location = translation_vector

            obj.select_set(False)
        else:
            print(f"Object {key} not found in the scene.")

# Test the function
if __name__ == '__main__':
    file_no = 4  # Set this to the desired file number
    app(file_no)
```

```
PYTHON INTERACTIVE CONSOLE 3.10.8 (main, Oct 18 2022, 21:01:35) [MSC v.1928 64 bit (AMD64)]

Builtin Modules:       bpy, bpy.data, bpy.ops, bpy.props, bpy.types, bpy.context, bpy.utils, bgl, gpu, blf, mathutils
Convenience Imports:   from mathutils import *; from math import *
Convenience Variables: C = bpy.context, D = bpy.data

>>>
36, -0.653424), (-0.453904, -0.470036, -0.756992)), orient_matrix_type='VIEW', mirror=False, use_proportional_edit=False, proportional_edit_falloff='SMOOTH', proportional_size=1, use_proportional_connected=False, use_proportional_projected=False, snap=False, snap_elements={'INCREMENT'}, use_snap_project=False, snap_target='CLOSEST', use_snap_self=True, use_snap_edit=True, use_snap_nonedit=True, use_snap_selectable=False)
bpy.ops.transform.rotate(value=1.11313, orient_axis='Y', orient_type='GLOBAL', orient_matrix=((1, 0, 0), (0, 1, 0), (0, 0, 1)), orient_matrix_type='GLOBAL', constraint_axis=(False, True, False), mirror=False, use_proportional_edit=False, proportional_edit_falloff='SMOOTH', proportional_size=1, use_proportional_connected=False, use_proportional_projected=False, snap=False, snap_elements={'INCREMENT'}, use_snap_project=False, snap_target='CLOSEST', use_snap_self=True, use_snap_edit=True, use_snap_nonedit=True, use_snap_selectable=False)
bpy.ops.transform.rotate(value=0.260589, orient_axis='X', orient_type='GLOBAL', orient_matrix=((1, 0, 0), (0, 1, 0), (0, 0, 1)), orient_matrix_type='GLOBAL', constraint_axis=(True, False, False), mirror=False, use_proportional_edit=False, proportional_edit_falloff='SMOOTH', proportional_size=1, use_proportional_connected=False, use_proportional_projected=False, snap=False, snap_elements={'INCREMENT'}, use_snap_project=False, snap_target='CLOSEST', use_snap_self=True, use_snap_edit=True, use_snap_nonedit=True, use_snap_selectable=False)
```

Figure 3: 5th keyframe element from roper.all rendered in blender

$$R = \begin{bmatrix} 0.98630136698630136 & -0.064579256360007827 & -0.15655577299412915 \\ 0.07632093933463796 & 0.99412915851272011 & 0.086105675146771003 \\ 0.14872798434442227 & -0.099804305283757333 & 0.98434442270058971 \end{bmatrix}$$

The last 3 values in .vue transform correspond to translation values.

$$\mathbf{v} = \begin{bmatrix} 0.002 & 52.764 & 41.211 \end{bmatrix}$$

Key frame editor source code has hard-coded reference points prim names which represent body parts. These reference points are pelvis and lfoot. If pelvis body part is present in .sex file then the code applies math operation to recenter body parts keeping pelvis as a center point. The code looks as follows:

```
      if(memcmp(prim_names[i], "lfoot" ,5) == 0 )
      {
        re_center=1;
        its_human=1;
        for (SLONG j = po->StartPoint; j < po->EndPoint; j++)
        {
          ASSERT(WITHIN(j, 1, next_prim_point - 1));

          y_centre += prim_points[j].Y;
        }
        y_centre /= po->EndPoint - po->StartPoint;
        y_centre -=FOOT_HEIGHT;
      }

      if(memcmp(prim_names[i], "pelvis" ,5) == 0 )
      {
        for (SLONG j = po->StartPoint; j < po->EndPoint; j++)
        {
          ASSERT(WITHIN(j, 1, next_prim_point - 1));

          x_centre += prim_points[j].X;
          z_centre += prim_points[j].Z;
//          y_centre += prim_points[j].Y;
        }
        x_centre /= po->EndPoint - po->StartPoint;
        z_centre /= po->EndPoint - po->StartPoint;
//        y_centre /= po->EndPoint - po->StartPoint;

        re_center=1;
        its_human=1;
      }
```

Listing 3: prim_edit.cpp read_multi_sex function

The code above calculates an average x and z center points for the pelvis by summing all its x-z vertices and then dividing by the vertices count.

roper.SEX PELVIS00 values are:

- Vertex: (-0.9545, 56.4253, 45.8323)
- Vertex: (-0.8031, 60.3635, 42.1830)
- Vertex: (-7.6487, 58.4347, 41.7473)
- Vertex: (-7.4077, 51.4368, 41.8131)
- Vertex: (6.3584, 50.9643, 41.8622)
- Vertex: (5.9993, 58.2492, 41.8826)
- Vertex: (-0.7547, 62.6026, 34.7955)
- Vertex: (-9.2669, 58.9335, 34.1141)
- Vertex: (-9.0106, 51.1023, 33.5794)
- Vertex: (-0.8406, 54.8909, 35.5727)
- Vertex: (7.5762, 50.5224, 33.6321)
- Vertex: (7.9547, 58.6220, 34.1551)
- Vertex: (1.6216, 48.4025, 41.9887)
- Vertex: (2.8132, 47.0871, 34.3781)
- Vertex: (-3.2558, 48.6338, 41.9548)
- Vertex: (-4.8130, 47.3536, 34.3283)
- Vertex: (1.0193, 49.1641, 35.5909)

- Vertex: (-2.9065, 49.2222, 35.5599)

These values are multiplied first by 2.56 and rounded to the nearest integer:

- Vertex: (-2, 144, 117)
- Vertex: (-2, 154, 108)
- Vertex: (-20, 150, 107)
- Vertex: (-19, 132, 107)
- Vertex: (16, 130, 107)
- Vertex: (15, 149, 107)
- Vertex: (-2, 160, 89)
- Vertex: (-24, 151, 87)
- Vertex: (-23, 131, 86)
- Vertex: (-2, 140, 91)
- Vertex: (19, 129, 86)
- Vertex: (20, 150, 87)
- Vertex: (4, 124, 107)
- Vertex: (7, 120, 88)
- Vertex: (-8, 124, 107)
- Vertex: (-12, 121, 88)
- Vertex: (3, 126, 91)
- Vertex: (-7, 126, 91)

Then the sums of the x and z vertices are calculated:

$$\begin{aligned}\sum_{i=1}^{18} x_i = & -2 + (-2) + (-20) + (-19) + 16 + 15 + (-2) \\ & + (-24) + (-23) + (-2) + 19 + 20 + 4 + 7 \\ & + (-8) + (-12) + 3 + (-7) = -34 \\ \sum_{i=1}^{18} z_i = & 117 + 108 + 107 + 107 + 107 + 107 + 89 \\ & + 87 + 86 + 91 + 86 + 87 + 107 + 88 \\ & + 107 + 88 + 91 + 91 = 1747\end{aligned}$$

Finally they are divded by the number of vertices and floored to the nearest integer: The average x and z values are given by:

$$\overline{X} = \frac{\sum_{i=1}^{18} x_i}{n} = \frac{-34}{18} \approx -1.88$$

$$\overline{Z} = \frac{\sum_{i=1}^{18} z_i}{n} = \frac{1747}{18} \approx 97.05$$