

Implementation of a 1D mass-spring model

- Basic implementation demonstration
- Elastic and damping force computation
- Forward Euler Method
- Automatic determination of time-step limits
- Introduction to implicit methods

Implementation of a 1D mass-spring model

- Basic implementation demonstration
- Elastic and damping force computation
- Forward Euler Method
- Automatic determination of time-step limits
- Introduction to implicit methods
- **WARNING: SOURCE CODE AHEAD!!**

```

#include <PhysBAM_Tools/Log/LOG.h>
#include <PhysBAM_Tools/Parsing/STRING_UTILITIES.h>
#include <PhysBAM_Tools/Read_Write/Utilities/FILE_UTILITIES.h>
#include <PhysBAM_Geometry/Geometry_Particles/GEOMETRY_PARTICLES.h>
#include <PhysBAM_Geometry/Geometry_Particles/REGISTER_GEOMETRY_READ_WRITE.h>
#include <PhysBAM_Geometry/Solids_Geometry/DEFORMABLE_GEOMETRY_COLLECTION.h>
#include <PhysBAM_Geometry/Topology_Based_Geometry/SEGMENTED_CURVE.h>
using namespace PhysBAM;

int main(int argc, char* argv[])
{
    typedef float T;
    typedef float RW;

    RW rw=RW();STREAM_TYPE stream_type(rw);
    typedef VECTOR<T,3> TV;

    LOG::Initialize_Logging();
    Initialize_Geometry_Particle();Initialize_Read_Write_Structures();

    const int n=11;                // Number of particles in wire mesh

    GEOMETRY_PARTICLES<TV> particles;
    SEGMENTED_CURVE<TV>& wire_curve=*SEGMENTED_CURVE<TV>::Create(particles);
    wire_curve.mesh.Initialize_Straight_Mesh(n);particles.array_collection->Add_Elements(n);
    for(int p=1;p<=n;p++) particles.X(p)=TV(0,(T)(1-p)/(T)(n-1),.5);

    DEFORMABLE_GEOMETRY_COLLECTION<TV> collection(particles);
    collection.Add_Structure(&wire_curve);

    FILE_UTILITIES::Create_Directory("output/0");collection.Write(stream_type,"output",0,0,true);

    LOG::Finish_Logging();
}

```

```

#include <PhysBAM_Tools/Log/LOG.h>
#include <PhysBAM_Tools/Parsing/STRING_UTILITIES.h>
#include <PhysBAM_Tools/Read_Write/Utilities/FILE_UTILITIES.h>
#include <PhysBAM_Geometry/Geometry_Particles/GEOMETRY_PARTICLES.h>
#include <PhysBAM_Geometry/Geometry_Particles/REGISTER_GEOMETRY_READ_WRITE.h>
#include <PhysBAM_Geometry/Solids_Geometry/DEFORMABLE_GEOMETRY_COLLECTION.h>
#include <PhysBAM_Geometry/Topology_Based_Geometry/SEGMENTED_CURVE.h>
using namespace PhysBAM;

int main(int argc, char* argv[])
{
    typedef float T;
    typedef float RW;

    RW rw=RW();STREAM_TYPE stream_type(rw);
    typedef VECTOR<T,3> TV;

    LOG::Initialize_Logging();
    Initialize_Geometry_Particle();Initialize_Read_Write_Structures();

    const int n=11;                // Number of particles in wire mesh

    GEOMETRY_PARTICLES<TV> particles;
    SEGMENTED_CURVE<TV>& wire_curve=*SEGMENTED_CURVE<TV>::Create(particles);
    wire_curve.mesh.Initialize_Straight_Mesh(n);particles.array_collection->Add_Elements(n);
    for(int p=1;p<=n;p++) particles.X(p)=TV(0,(T)(1-p)/(T)(n-1),.5);

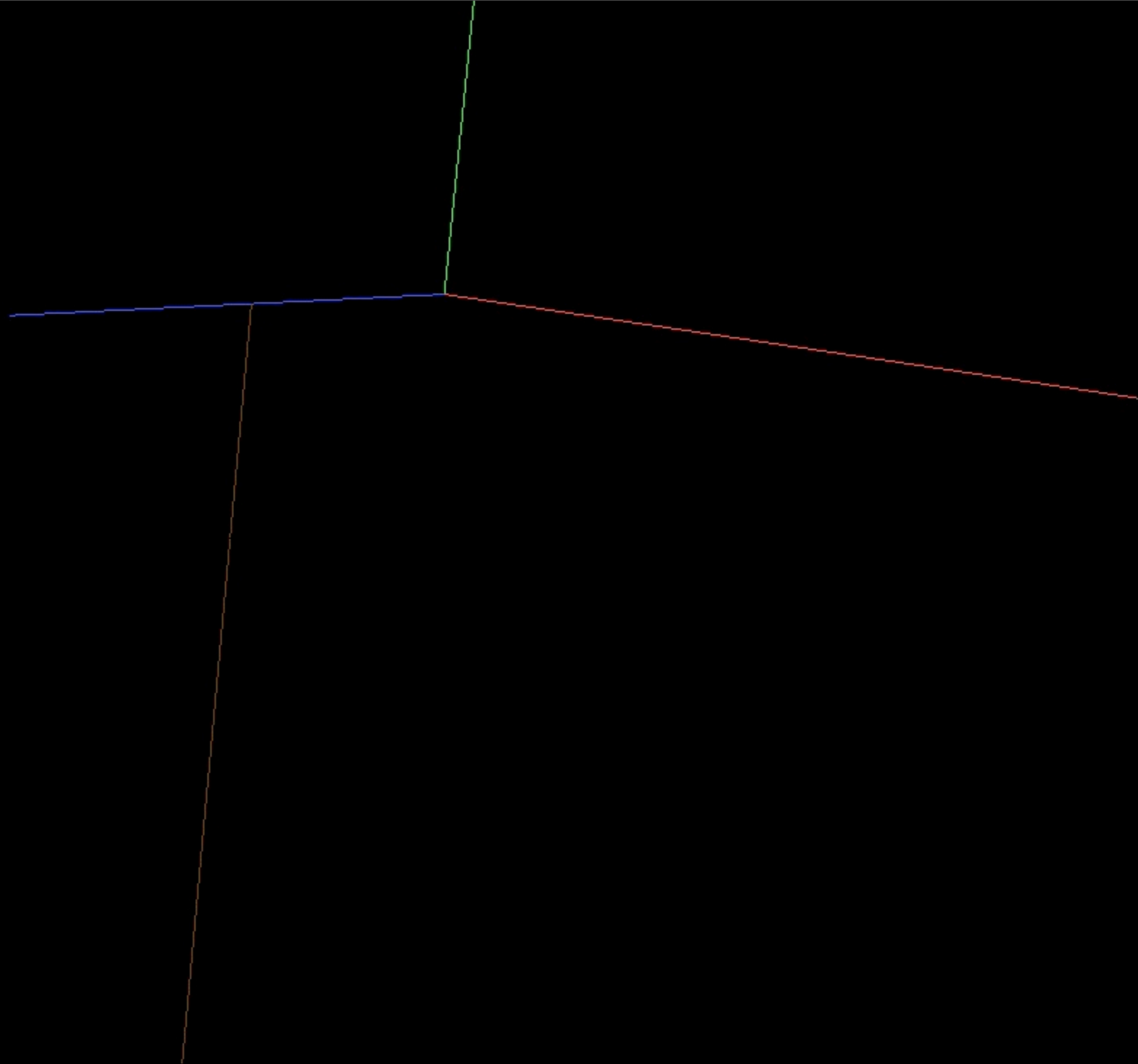
    DEFORMABLE_GEOMETRY_COLLECTION<TV> collection(particles);
    collection.Add_Structure(&wire_curve);

    FILE_UTILITIES::Create_Directory("output/0");collection.Write(stream_type,"output",0,0,true);

    LOG::Finish_Logging();
}

```

0fps
frame 0



```

#include <PhysBAM_Geometry/Topology_Based_Geometry/SEGMENTED_CURVE.h>
#include <PhysBAM_Geometry/Topology_Based_Geometry/FREE_PARTICLES.h>
using namespace PhysBAM;

int main(int argc, char* argv[])
{
[... ]

    const int n=11;                // Number of particles in wire mesh
    const int number_of_frames=100; // Total number of frames
    const T frame_time=.05;        // Frame (snapshot) interval

    GEOMETRY_PARTICLES<TV> particles; particles.Store_Velocity();
    SEGMENTED_CURVE<TV>& wire_curve=*SEGMENTED_CURVE<TV>::Create(particles);
    wire_curve.mesh.Initialize_Straight_Mesh(n); particles.array_collection->Add_Elements(n);
    for(int p=1;p<=n;p++) particles.X(p)=TV(0,(T)(1-p)/(T)(n-1),.5);
    FREE_PARTICLES<TV>& wire_particles=*FREE_PARTICLES<TV>::Create(particles);
    for(int p=1;p<=n;p++) wire_particles.nodes.Append(p);

    DEFORMABLE_GEOMETRY_COLLECTION<TV> collection(particles);
    collection.Add_Structure(&wire_curve); collection.Add_Structure(&wire_particles);

    T dt_max=.001, dt;
    T time=0.;

    FILE_UTILITIES::Create_Directory("output/0"); collection.Write(stream_type,"output",0,0,true);

[... ]

    LOG::Finish_Logging();
}

```

```

#include <PhysBAM_Geometry/Topology_Based_Geometry/SEGMENTED_CURVE.h>
#include <PhysBAM_Geometry/Topology_Based_Geometry/FREE_PARTICLES.h>
using namespace PhysBAM;

int main(int argc, char* argv[])
{
[... ]

    const int n=11;                // Number of particles in wire mesh
    const int number_of_frames=100; // Total number of frames
    const T frame_time=.05;        // Frame (snapshot) interval

    GEOMETRY_PARTICLES<TV> particles; particles.Store_Velocity();
    SEGMENTED_CURVE<TV>& wire_curve=*SEGMENTED_CURVE<TV>::Create(particles);
    wire_curve.mesh.Initialize_Straight_Mesh(n); particles.array_collection->Add_Elements(n);
    for(int p=1;p<=n;p++) particles.X(p)=TV(0,(T)(1-p)/(T)(n-1),.5);
    FREE_PARTICLES<TV>& wire_particles=*FREE_PARTICLES<TV>::Create(particles);
    for(int p=1;p<=n;p++) wire_particles.nodes.Append(p);

    DEFORMABLE_GEOMETRY_COLLECTION<TV> collection(particles);
    collection.Add_Structure(&wire_curve); collection.Add_Structure(&wire_particles);

    T dt_max=.001, dt;
    T time=0.;

    FILE_UTILITIES::Create_Directory("output/0"); collection.Write(stream_type,"output",0,0,true);

[... ]

    LOG::Finish_Logging();
}

```

```

#include <PhysBAM_Geometry/Topology_Based_Geometry/SEGMENTED_CURVE.h>
#include <PhysBAM_Geometry/Topology_Based_Geometry/FREE_PARTICLES.h>
using namespace PhysBAM;

int main(int argc, char* argv[])
{
    [...]

    FILE_UTILITIES::Create_Directory("output/0"); collection.Write(stream_type, "output", 0, 0, true);

    for(int frame=1; frame<=number_of_frames; frame++){
        T frame_end_time=frame_time*(T)frame;

        for(; time<frame_end_time; time+=dt){
            dt=std::min(dt_max, (T)1.001*(frame_end_time-time));

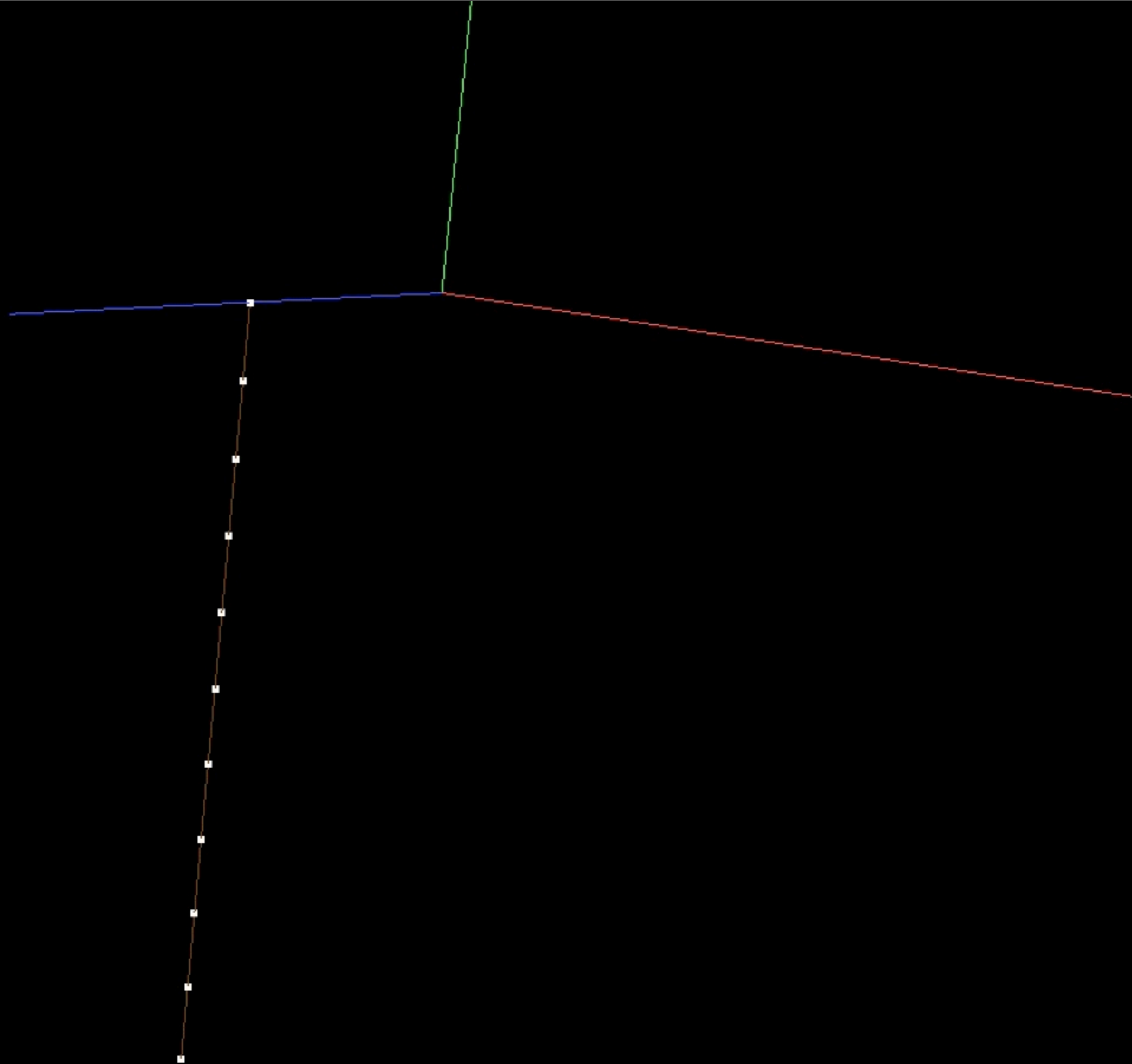
            // Set upper endpoint position and velocity
            T angular_velocity=two_pi/(frame_time*(T)number_of_frames);
            particles.X(1)=TV(.5*sin(time*angular_velocity), 0, .5*cos(time*angular_velocity));
            particles.V(1)=TV(.5*angular_velocity*cos(time*angular_velocity), 0,
                -.5*angular_velocity*sin(time*angular_velocity));

            FILE_UTILITIES::Create_Directory("output/"+STRING_UTILITIES::Value_To_String(frame));
            collection.Write(stream_type, "output", frame, 0, true);
        }
    }

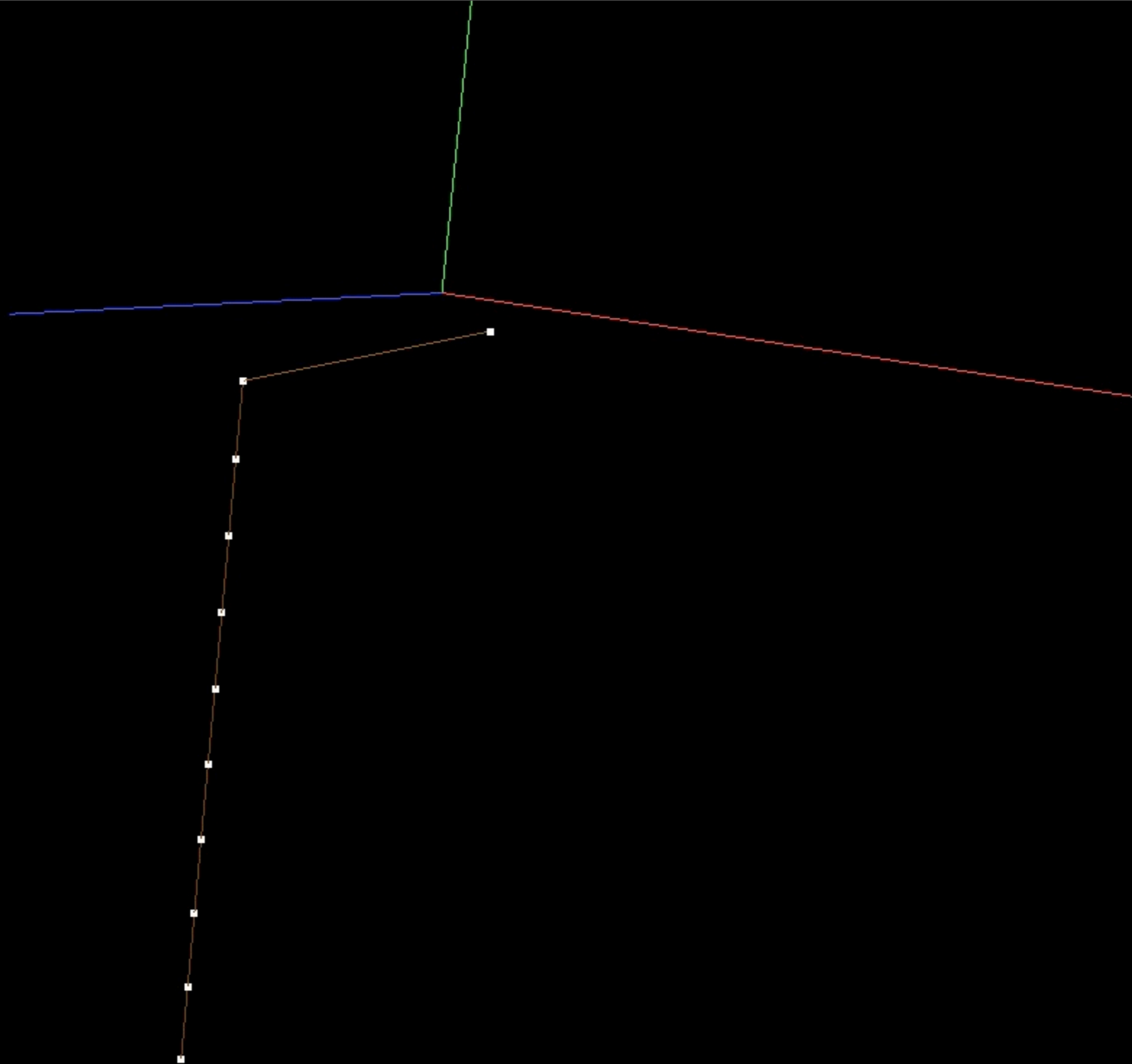
    LOG::Finish_Logging();
}

```

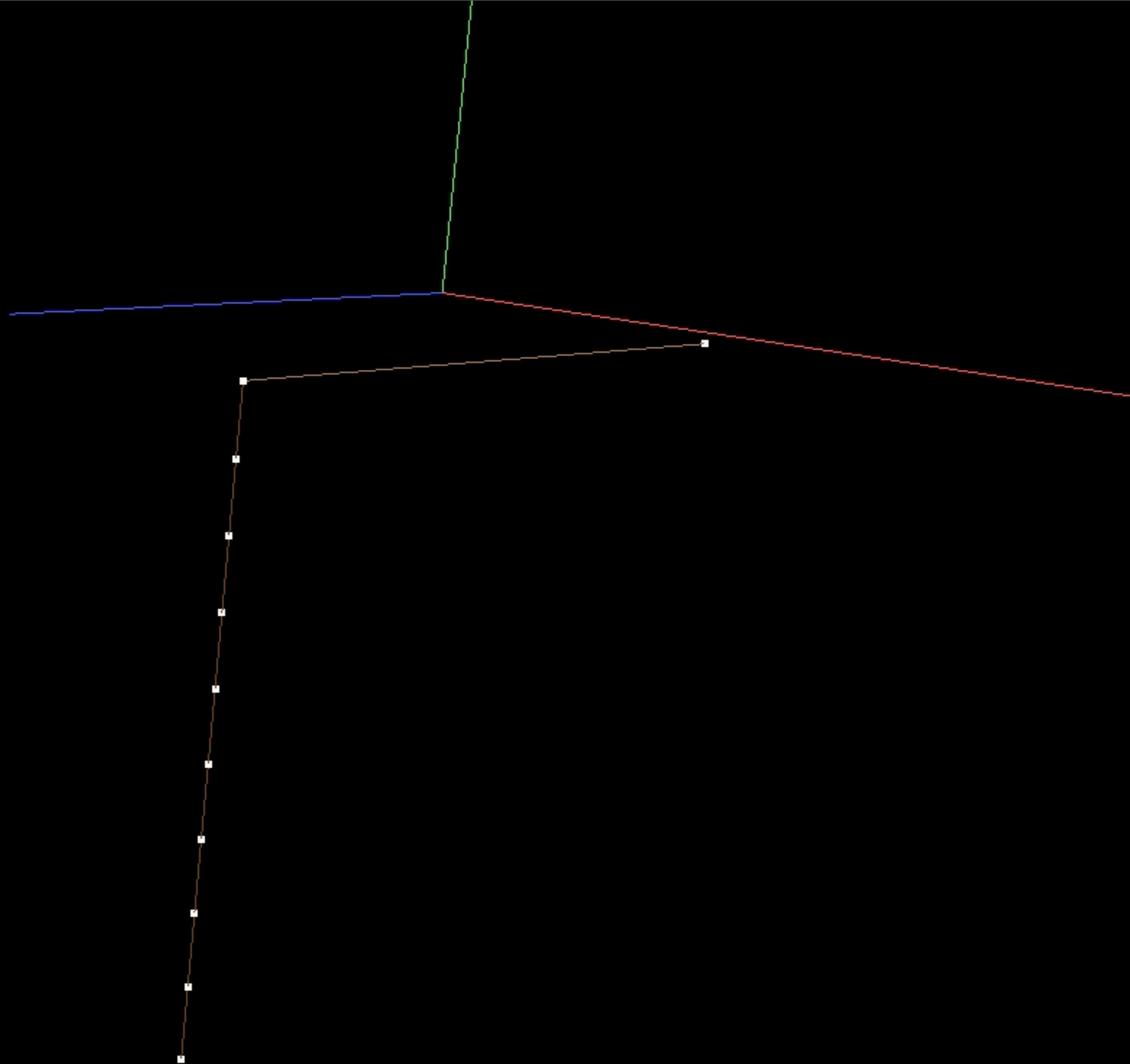

0fps
frame 0



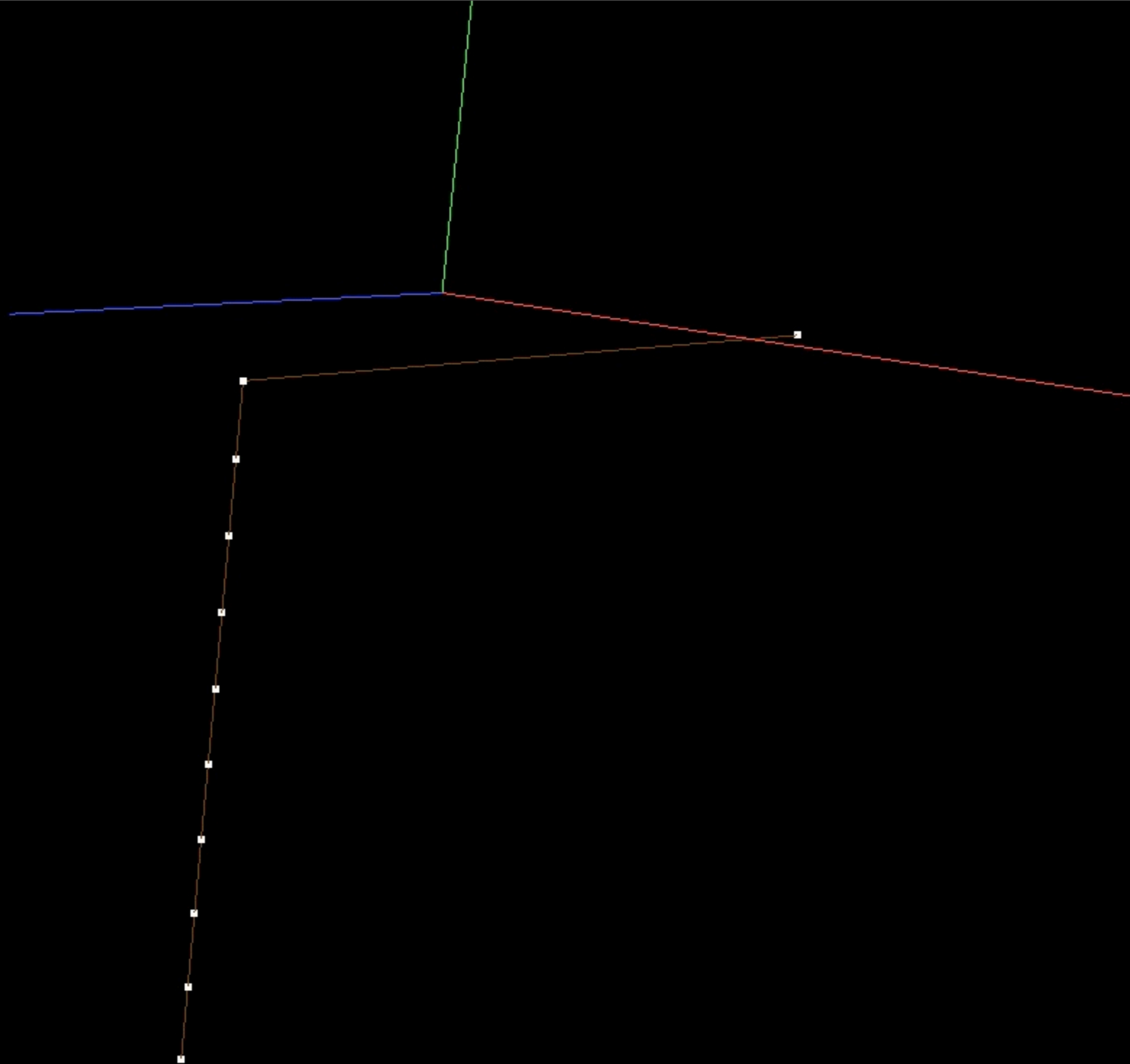
0fps
frame 10



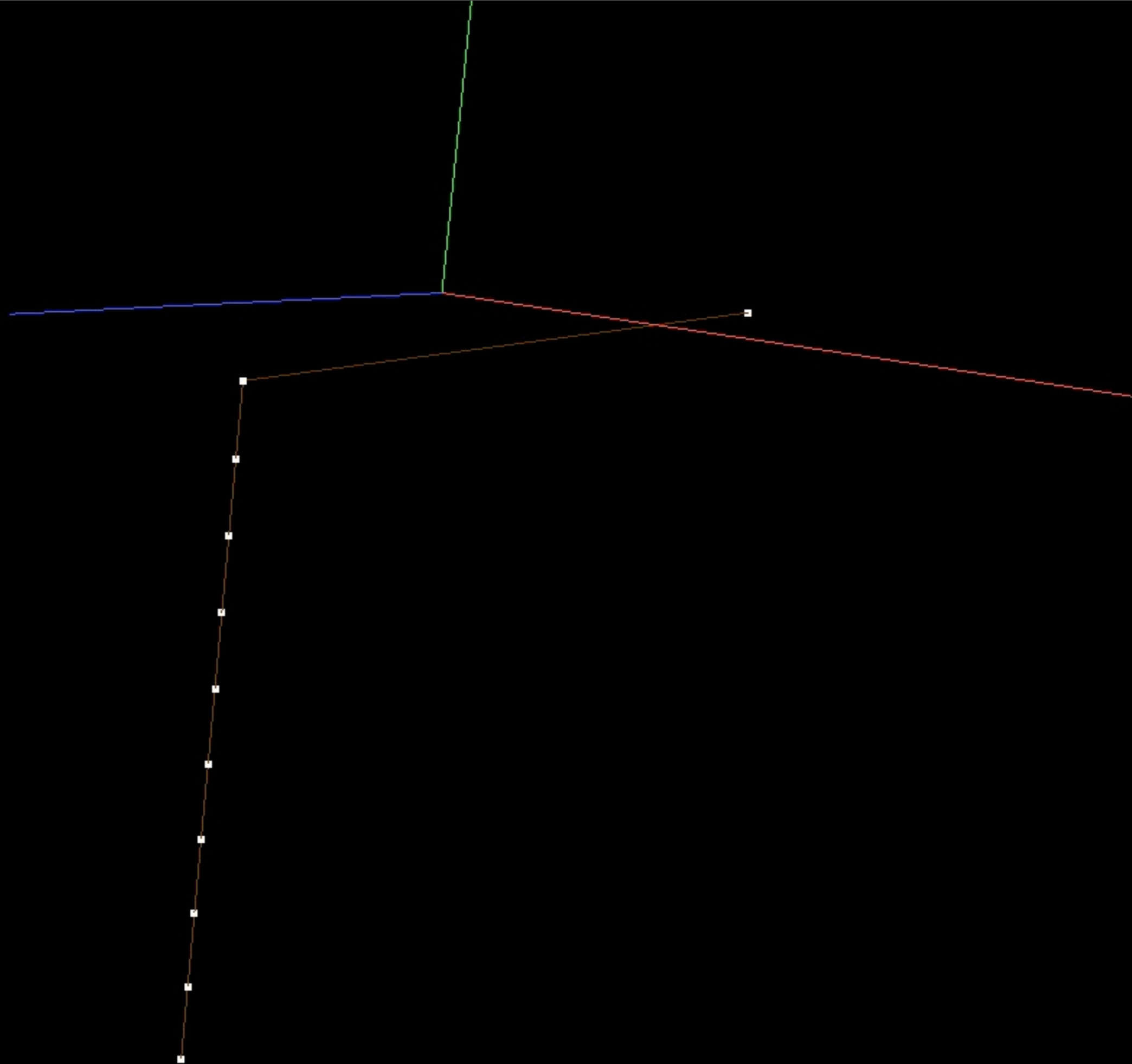
0fps
frame 20



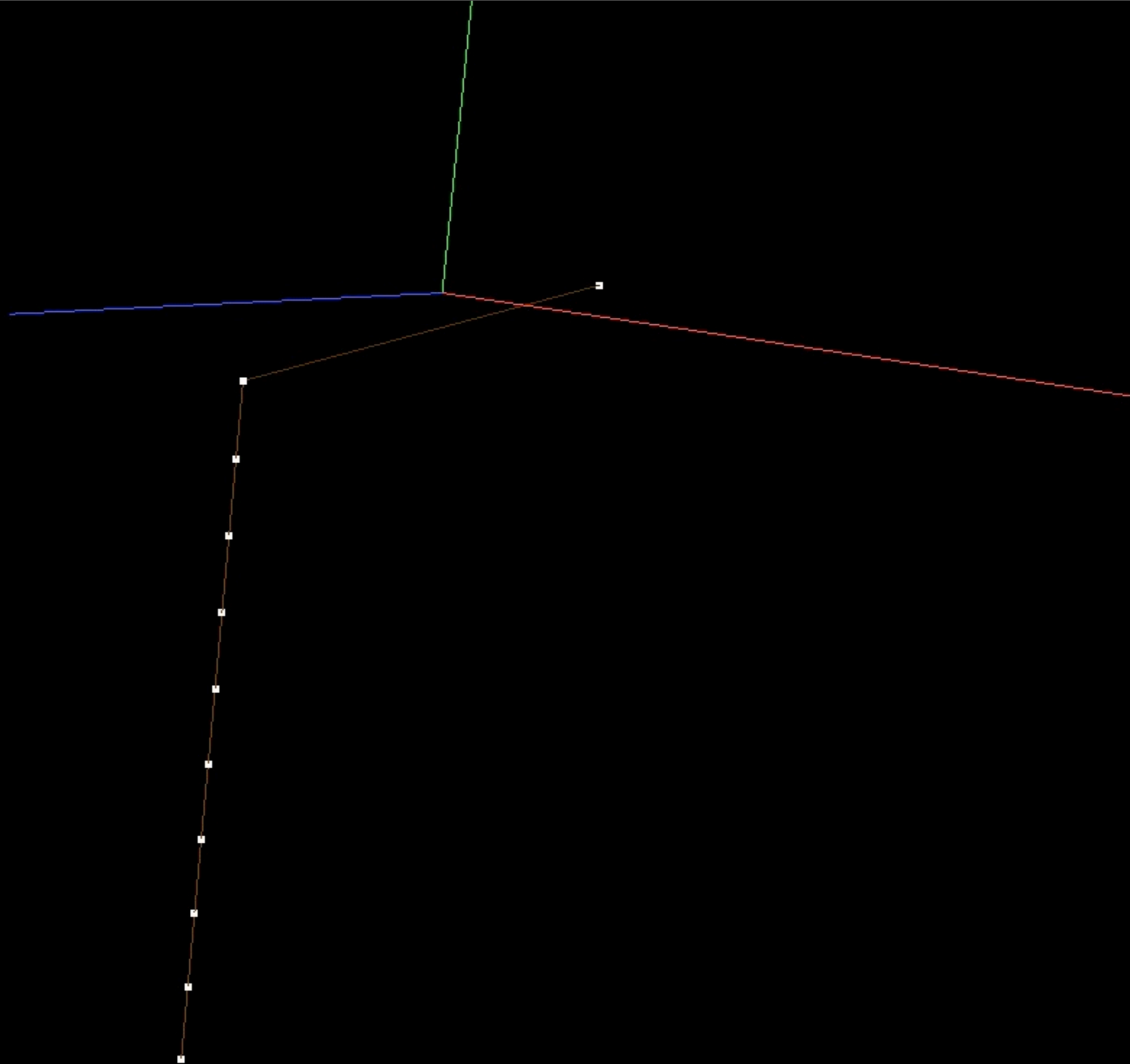
0fps
frame 30



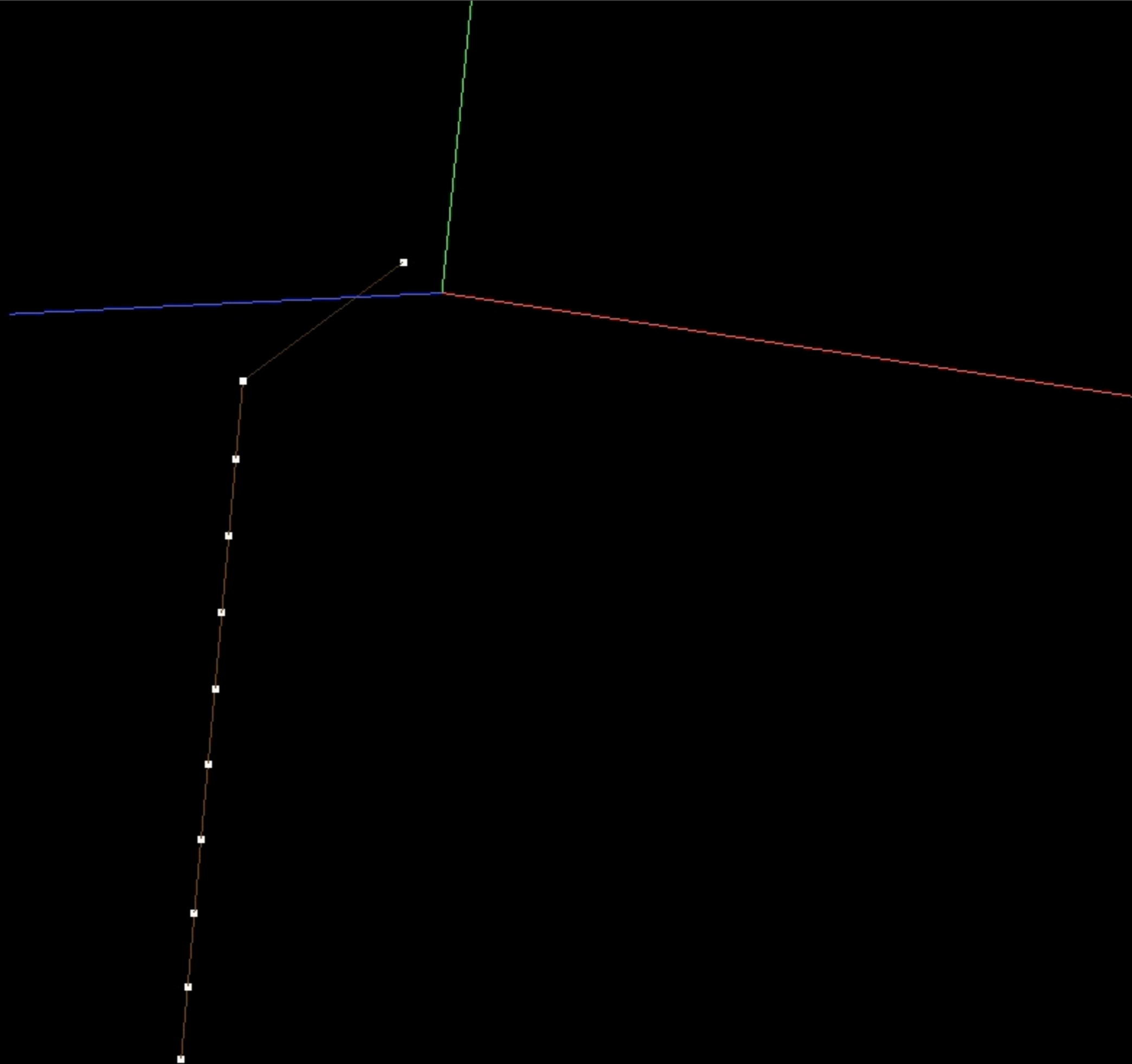
0fps
frame 40



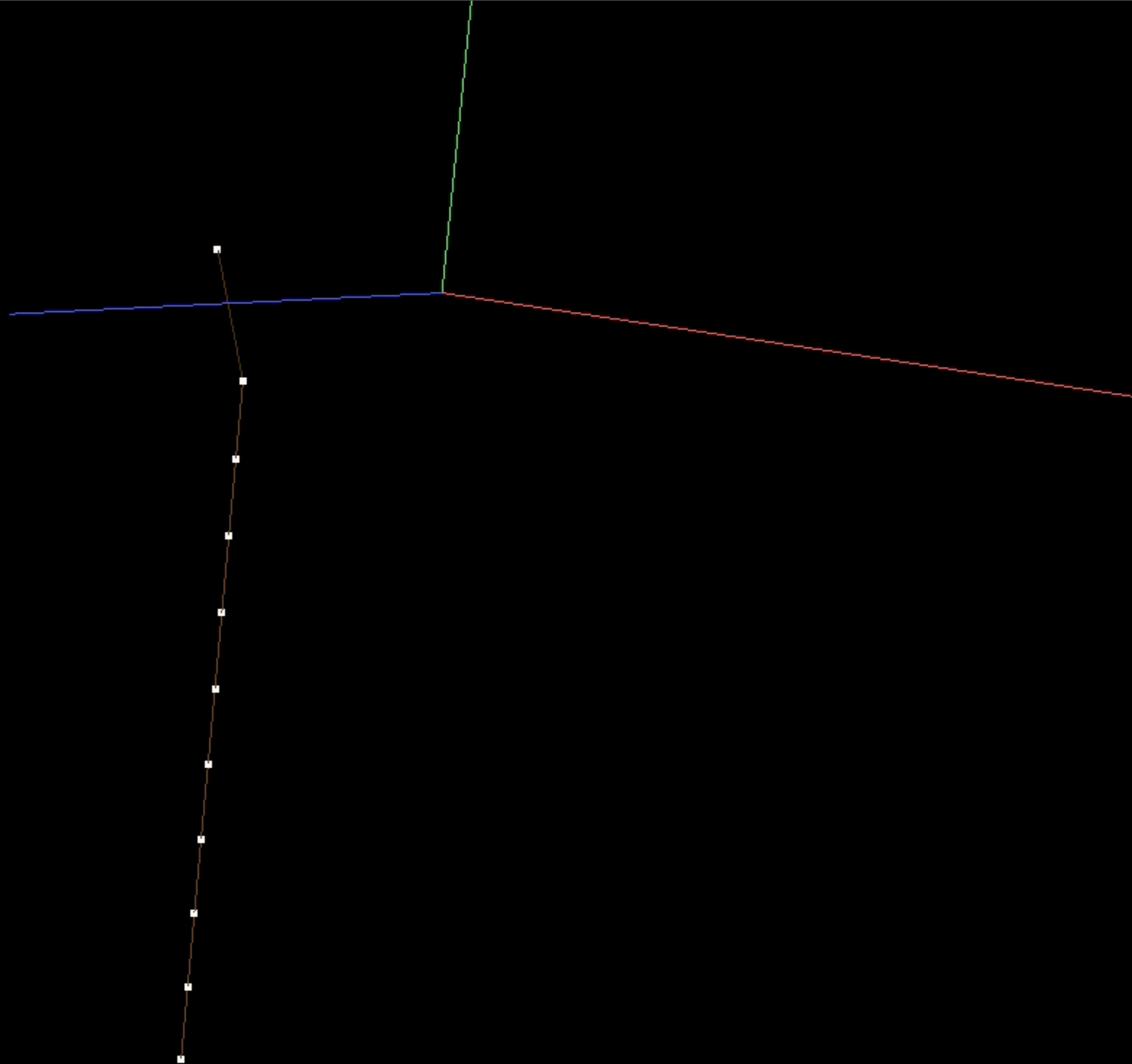
0fps
frame 50



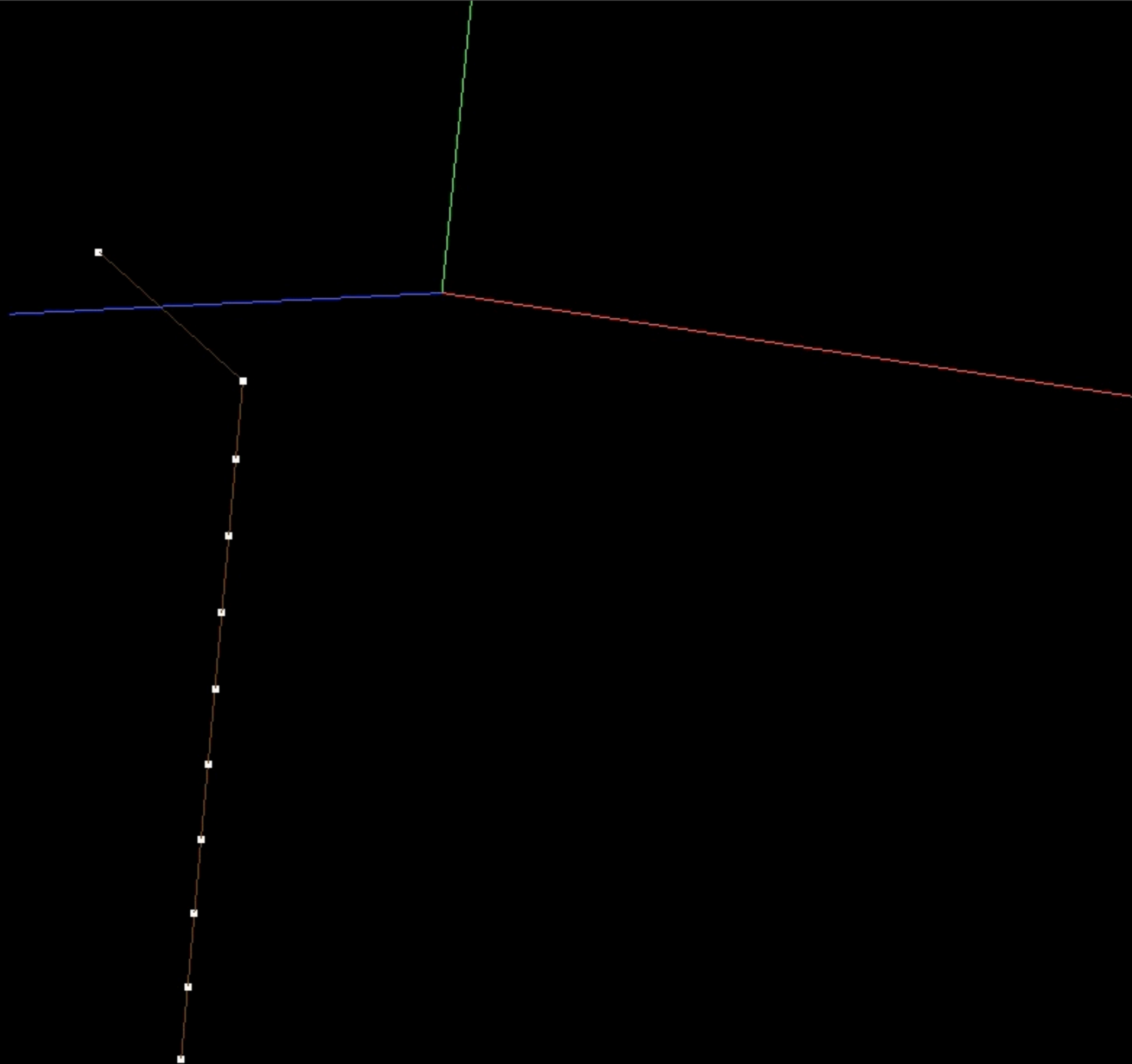
0fps
frame 60



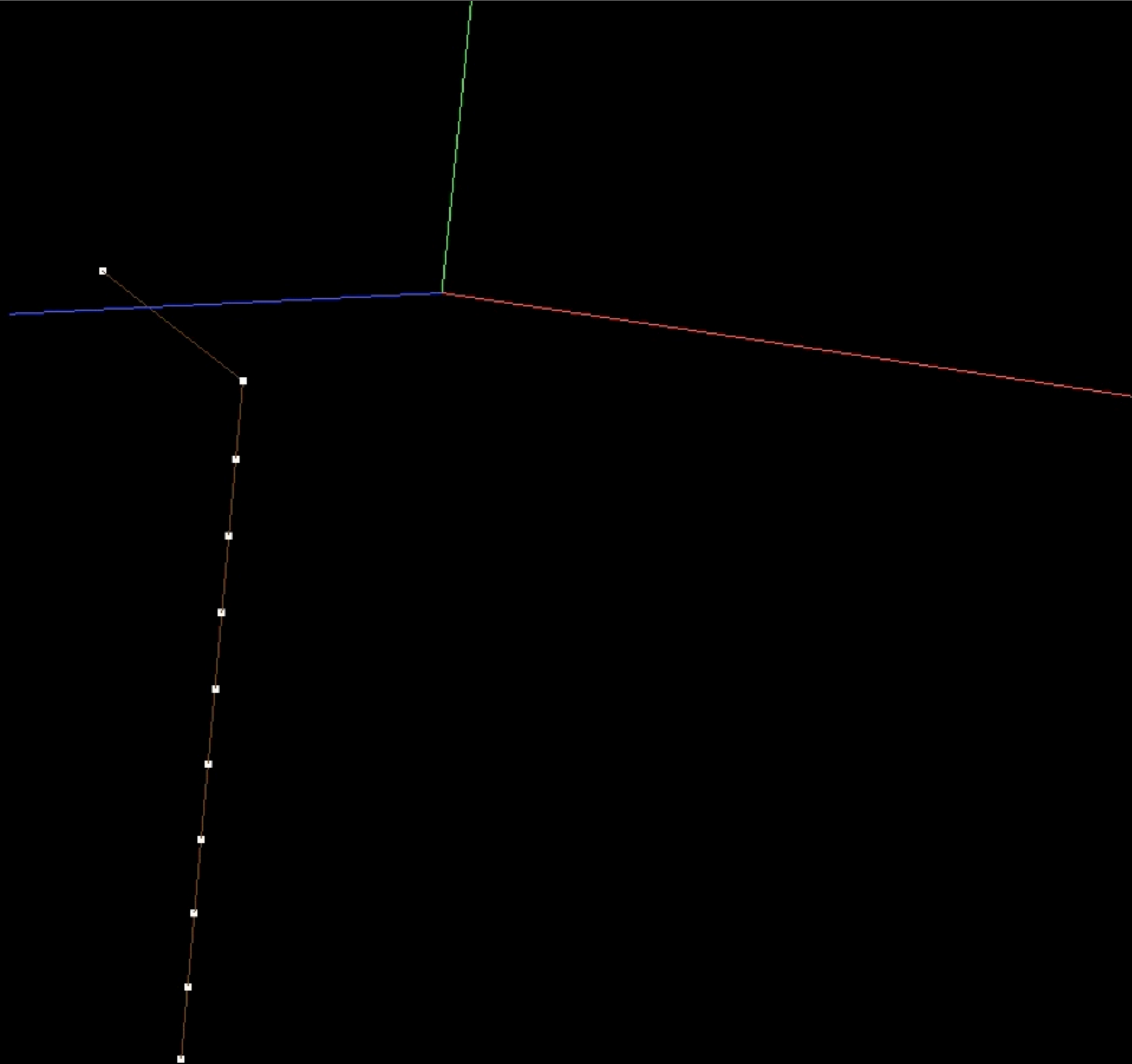
0fps
frame 70



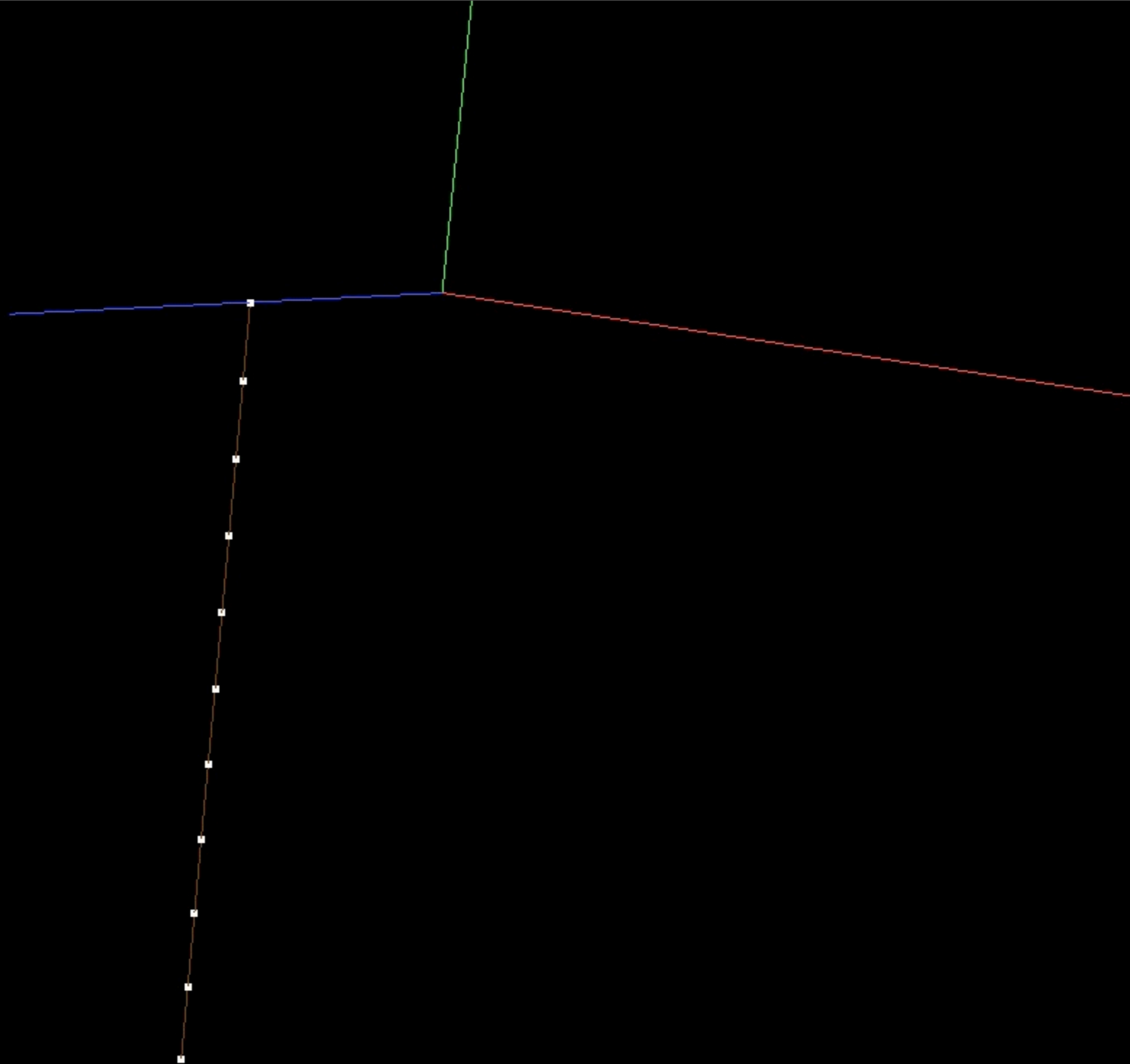
0fps
frame 80



0fps
frame 90



0fps
frame 100



[...]

```
LOG::Initialize_Logging();
Initialize_Geometry_Particle();Initialize_Read_Write_Structures();

const int n=11;                // Number of particles in wire mesh
const int number_of_frames=100; // Total number of frames
const T frame_time=.05;        // Frame (snapshot) interval
const T youngs_modulus=10.;    // Elasticity and damping coefficients
const T wire_length=1.;
const T restlength=wire_length/(T)(n-1); // Restlength (per each spring)

GEOMETRY_PARTICLES<TV> particles;particles.Store_Velocity();
SEGMENTED_CURVE<TV>& wire_curve=*SEGMENTED_CURVE<TV>::Create(particles);
wire_curve.mesh.Initialize_Straight_Mesh(n);particles.array_collection->Add_Elements(n);
for(int p=1;p<=n;p++) particles.X(p)=TV(0,(T)(1-p)/(T)(n-1),.5);
FREE_PARTICLES<TV>& wire_particles=*FREE_PARTICLES<TV>::Create(particles);
for(int p=1;p<=n;p++) wire_particles.nodes.Append(p);

DEFORMABLE_GEOMETRY_COLLECTION<TV> collection(particles);
collection.Add_Structure(&wire_curve);collection.Add_Structure(&wire_particles);

ARRAY<TV> force(n);

T dt_max=.001,dt;
T time=0.;

FILE_UTILITIES::Create_Directory("output/0");collection.Write(stream_type,"output",0,0,true);

for(int frame=1;frame<=number_of_frames;frame++){
    T frame_end_time=frame_time*(T)frame;

    for(;time<frame_end_time;time+=dt){
        dt=std::min(dt_max,(T)1.001*(frame_end_time-time));

        // Set upper endpoint position and velocity
```

[...]

```
LOG::Initialize_Logging();
Initialize_Geometry_Particle();Initialize_Read_Write_Structures();

const int n=11;                // Number of particles in wire mesh
const int number_of_frames=100; // Total number of frames
const T frame_time=.05;        // Frame (snapshot) interval
const T youngs_modulus=10.;    // Elasticity and damping coefficients
const T wire_length=1.;
const T restlength=wire_length/(T)(n-1); // Restlength (per each spring)

GEOMETRY_PARTICLES<TV> particles;particles.Store_Velocity();
SEGMENTED_CURVE<TV>& wire_curve=*SEGMENTED_CURVE<TV>::Create(particles);
wire_curve.mesh.Initialize_Straight_Mesh(n);particles.array_collection->Add_Elements(n);
for(int p=1;p<=n;p++) particles.X(p)=TV(0,(T)(1-p)/(T)(n-1),.5);
FREE_PARTICLES<TV>& wire_particles=*FREE_PARTICLES<TV>::Create(particles);
for(int p=1;p<=n;p++) wire_particles.nodes.Append(p);

DEFORMABLE_GEOMETRY_COLLECTION<TV> collection(particles);
collection.Add_Structure(&wire_curve);collection.Add_Structure(&wire_particles);

ARRAY<TV> force(n);

T dt_max=.001,dt;
T time=0.;

FILE_UTILITIES::Create_Directory("output/0");collection.Write(stream_type,"output",0,0,true);

for(int frame=1;frame<=number_of_frames;frame++){
    T frame_end_time=frame_time*(T)frame;

    for(;time<frame_end_time;time+=dt){
        dt=std::min(dt_max,(T)1.001*(frame_end_time-time));

        // Set upper endpoint position and velocity
```

```
[...]
```

```
for(int frame=1;frame<=number_of_frames;frame++){
    T frame_end_time=frame_time*(T)frame;

    for(;time<frame_end_time;time+=dt){
        dt=std::min(dt_max,(T)1.001*(frame_end_time-time));

        // Set upper endpoint position and velocity
        T angular_velocity=two_pi/(frame_time*(T)number_of_frames);
        particles.X(1)=TV(.5*sin(time*angular_velocity),0,.5*cos(time*angular_velocity));
        particles.V(1)=TV(.5*angular_velocity*cos(time*angular_velocity),0,
            -.5*angular_velocity*sin(time*angular_velocity));

        force.Fill(TV()); // Clear all forces from previous iterations

        // Add spring force
        for(int s=1;s<=wire_curve.mesh.elements.m;s++){
            int p1,p2;wire_curve.mesh.elements(s).Get(p1,p2);
            TV X1=particles.X(p1),X2=particles.X(p2);
            TV normal=(X1-X2).Normalized();
            T length=(X1-X2).Magnitude();
            TV f=-normal*youngs_modulus*(length/restlength-1.);
            force(p1)+=f;force(p2)-=f;}

        FILE_UTILITIES::Create_Directory("output/"+STRING_UTILITIES::Value_To_String(frame));
        collection.Write(stream_type,"output",frame,0,true);
    }
}

LOG::Finish_Logging();
}
```

```

[... ]
const int n=11; // Number of particles in wire mesh
const int number_of_frames=100; // Total number of frames
const T frame_time=.05; // Frame (snapshot) interval
const T youngs_modulus=10.; // Elasticity and damping coefficients
const T damping_coefficient=10.;
const T wire_length=1.;
const T restlength=wire_length/(T)(n-1); // Restlength (per each spring)

```

```
[... ]
```

```

// Add spring force
for(int s=1;s<=wire_curve.mesh.elements.m;s++){
    int p1,p2;wire_curve.mesh.elements(s).Get(p1,p2);
    TV X1=particles.X(p1),X2=particles.X(p2);
    TV normal=(X1-X2).Normalized();
    T length=(X1-X2).Magnitude();
    TV f=-normal*youngs_modulus*(length/restlength-1.);
    force(p1)+=f;force(p2)-=f;}

```

```

// Add damping force
for(int s=1;s<=wire_curve.mesh.elements.m;s++){
    int p1,p2;wire_curve.mesh.elements(s).Get(p1,p2);
    TV X1=particles.X(p1),X2=particles.X(p2);
    TV V1=particles.V(p1),V2=particles.V(p2);
    TV normal=(X1-X2).Normalized();
    T vrel=TV::Dot_Product(normal,V1-V2);
    TV f=-damping_coefficient*vrel*normal;
    force(p1)+=f;force(p2)-=f;}

```

```

FILE_UTILITIES::Create_Directory("output/"+STRING_UTILITIES::Value_To_String(frame));
collection.Write(stream_type,"output",frame,0,true);

```

```
}
```

```
}
```

```
LOG::Finish_Logging();
```

```
}
```

[...]

```
const int n=11; // Number of particles in wire mesh
const int number_of_frames=100; // Total number of frames
const T frame_time=.05; // Frame (snapshot) interval
const T youngs_modulus=10.; // Elasticity and damping coefficients
const T damping_coefficient=10.;
const T wire_length=1.;
const T restlength=wire_length/(T)(n-1); // Restlength (per each spring)
```

[...]

```
// Add spring force
for(int s=1;s<=wire_curve.mesh.elements.m;s++){
    int p1,p2;wire_curve.mesh.elements(s).Get(p1,p2);
    TV X1=particles.X(p1),X2=particles.X(p2);
    TV normal=(X1-X2).Normalized();
    T length=(X1-X2).Magnitude();
    TV f=-normal*youngs_modulus*(length/restlength-1.);
    force(p1)+=f;force(p2)-=f;}
```

```
// Add damping force
for(int s=1;s<=wire_curve.mesh.elements.m;s++){
    int p1,p2;wire_curve.mesh.elements(s).Get(p1,p2);
    TV X1=particles.X(p1),X2=particles.X(p2);
    TV V1=particles.V(p1),V2=particles.V(p2);
    TV normal=(X1-X2).Normalized();
    T vrel=TV::Dot_Product(normal,V1-V2);
    TV f=-damping_coefficient*vrel*normal;
    force(p1)+=f;force(p2)-=f;}
```

```
FILE_UTILITIES::Create_Directory("output/"+STRING_UTILITIES::Value_To_String(frame));
collection.Write(stream_type,"output",frame,0,true);
```

```
}
```

```
}
```

```
LOG::Finish_Logging();
```

```
}
```



```

[... ]

const int n=11; // Number of particles in wire mesh
const int number_of_frames=100; // Total number of frames
const T frame_time=.05; // Frame (snapshot) interval
const T youngs_modulus=10.; // Elasticity and damping coefficients
const T damping_coefficient=10.;
const T wire_mass=1.; // Mass and length for entire wire
const T wire_length=1.;
const T mass=wire_mass/(T)n; // Mass (per each particle)
const T restlength=wire_length/(T)(n-1); // Restlength (per each spring)

[... ]

// Add damping force
for(int s=1;s<=wire_curve.mesh.elements.m;s++){
    int p1,p2;wire_curve.mesh.elements(s).Get(p1,p2);
    TV X1=particles.X(p1),X2=particles.X(p2);
    TV V1=particles.V(p1),V2=particles.V(p2);
    TV normal=(X1-X2).Normalized();
    T vrel=TV::Dot_Product(normal,V1-V2);
    TV f=-damping_coefficient*vrel*normal;
    force(p1)+=f;force(p2)-=f;}

// Add gravity
force+=-TV::Axis_Vector(2)*mass*9.81;

FILE_UTILITIES::Create_Directory("output/"+STRING_UTILITIES::Value_To_String(frame));
collection.Write(stream_type,"output",frame,0,true);
}
}

LOG::Finish_Logging();
}

```

[...]

```
const int n=11; // Number of particles in wire mesh
const int number_of_frames=100; // Total number of frames
const T frame_time=.05; // Frame (snapshot) interval
const T youngs_modulus=10.; // Elasticity and damping coefficients
const T damping_coefficient=10.;
const T wire_mass=1.; // Mass and length for entire wire
const T wire_length=1.;
const T mass=wire_mass/(T)n; // Mass (per each particle)
const T restlength=wire_length/(T)(n-1); // Restlength (per each spring)
```

[...]

```
// Add damping force
for(int s=1;s<=wire_curve.mesh.elements.m;s++){
    int p1,p2;wire_curve.mesh.elements(s).Get(p1,p2);
    TV X1=particles.X(p1),X2=particles.X(p2);
    TV V1=particles.V(p1),V2=particles.V(p2);
    TV normal=(X1-X2).Normalized();
    T vrel=TV::Dot_Product(normal,V1-V2);
    TV f=-damping_coefficient*vrel*normal;
    force(p1)+=f;force(p2)-=f;}
```

```
// Add gravity
force+=-TV::Axis_Vector(2)*mass*9.81;
```

```
FILE_UTILITIES::Create_Directory("output/"+STRING_UTILITIES::Value_To_String(frame));
collection.Write(stream_type,"output",frame,0,true);
```

```
}
```

```
}
```

```
LOG::Finish_Logging();
```

```
}
```

```

[... ]

ARRAY<TV> force(n),dX(n),dV(n);

T dt_max=.0001,dt;
T time=0.;

[... ]
    // Add gravity
    force+=-TV::Axis_Vector(2)*mass*9.81;

    // Apply the Forward Euler method
    dX=dt*particles.V; // Compute position change
    dV=(dt/mass)*force; // Compute velocity change
    dX(1)=dV(1)=TV(); // First particle has externally prescribed motion; do not alter
    particles.X+=dX; // Update particle positions and velocities
    particles.V+=dV;

    FILE_UTILITIES::Create_Directory("output/"+STRING_UTILITIES::Value_To_String(frame));
    collection.Write(stream_type,"output",frame,0,true);
}
}

LOG::Finish_Logging();
}

```

```

[... ]

ARRAY<TV> force(n),dX(n),dV(n);

T dt_max=.0001,dt;
T time=0.;

[... ]
    // Add gravity
    force+=-TV::Axis_Vector(2)*mass*9.81;

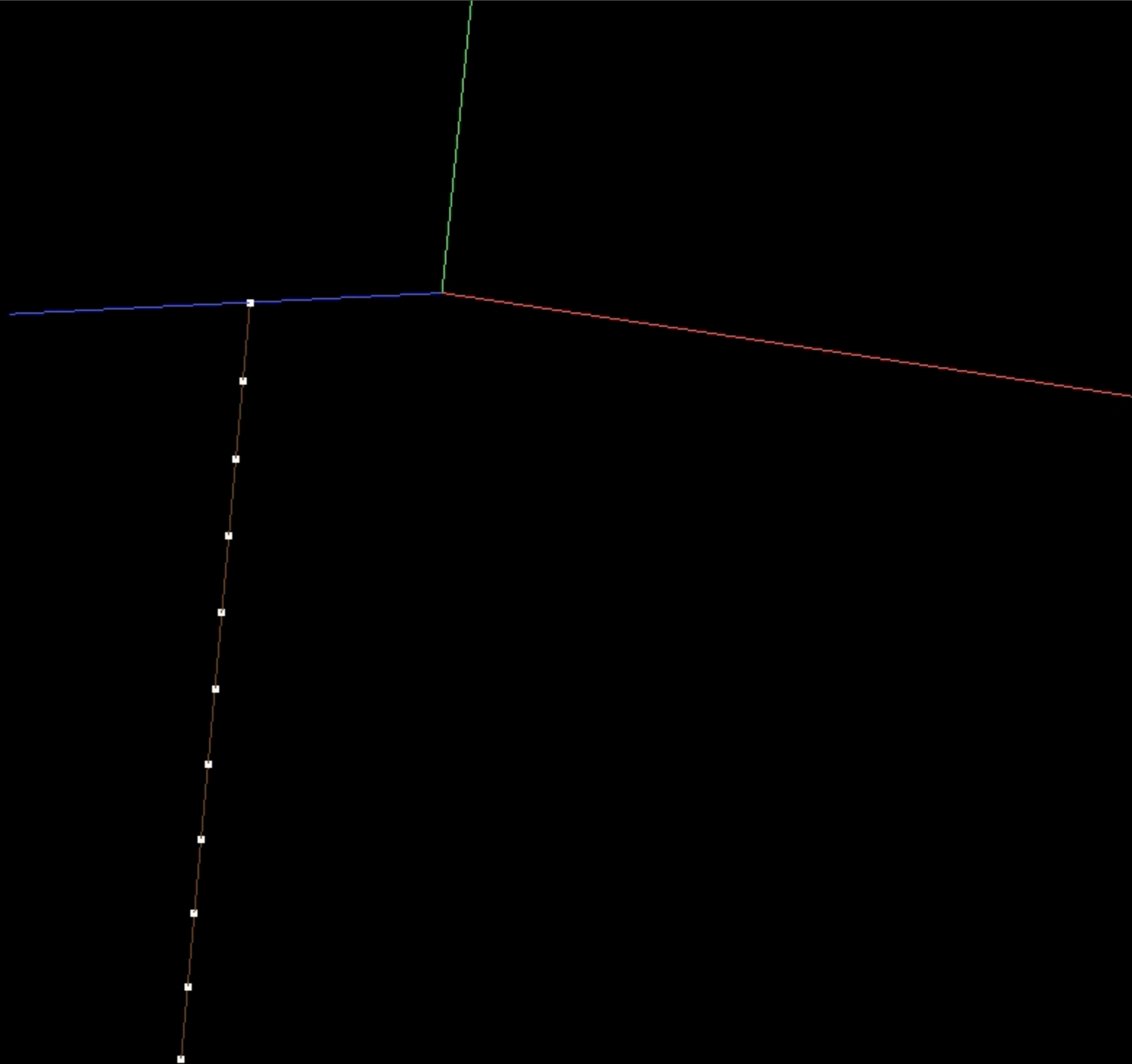
    // Apply the Forward Euler method
    dX=dt*particles.V; // Compute position change
    dV=(dt/mass)*force; // Compute velocity change
    dX(1)=dV(1)=TV(); // First particle has externally prescribed motion; do not alter
    particles.X+=dX; // Update particle positions and velocities
    particles.V+=dV;

    FILE_UTILITIES::Create_Directory("output/"+STRING_UTILITIES::Value_To_String(frame));
    collection.Write(stream_type,"output",frame,0,true);
}
}

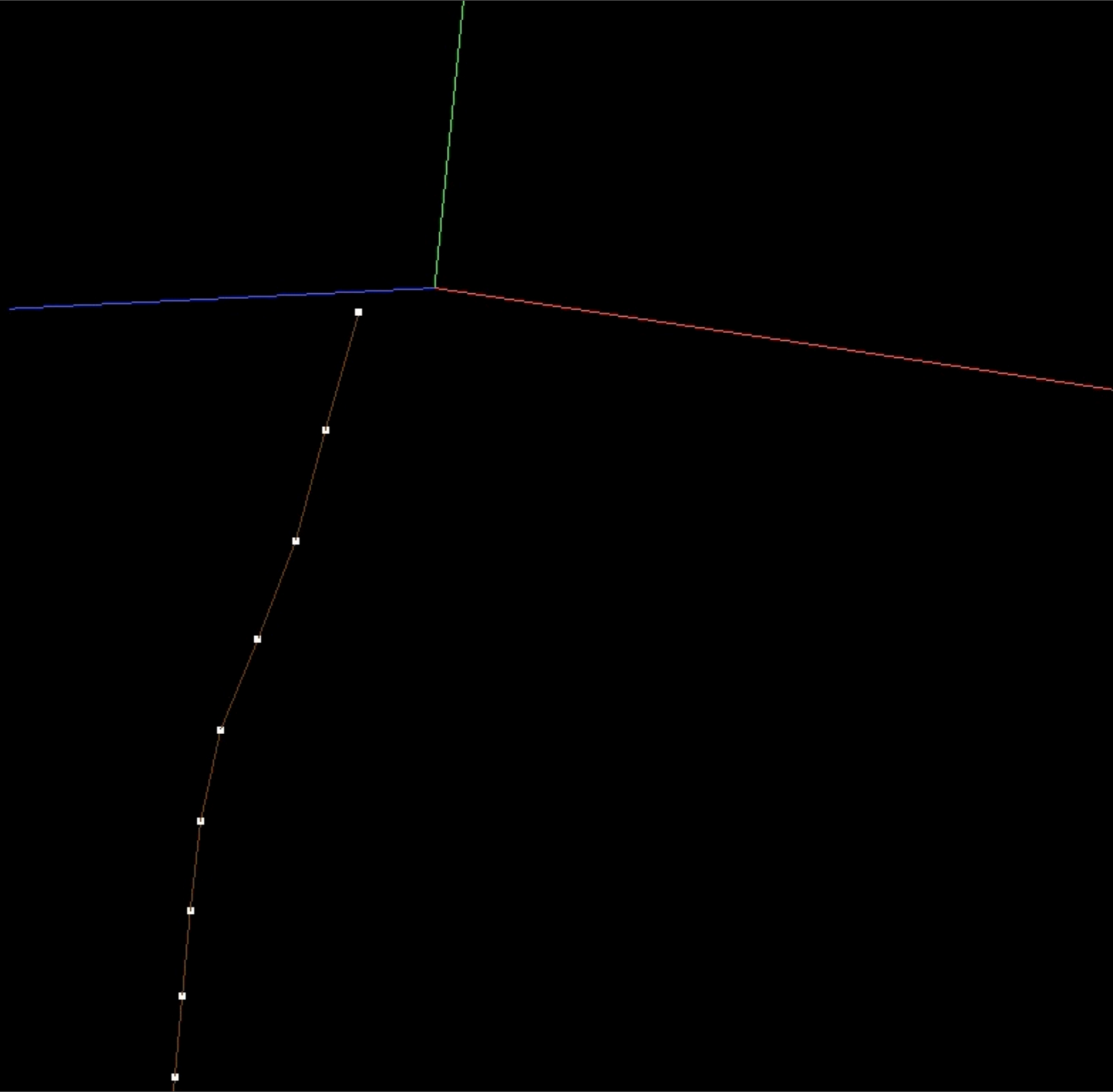
LOG::Finish_Logging();
}

```

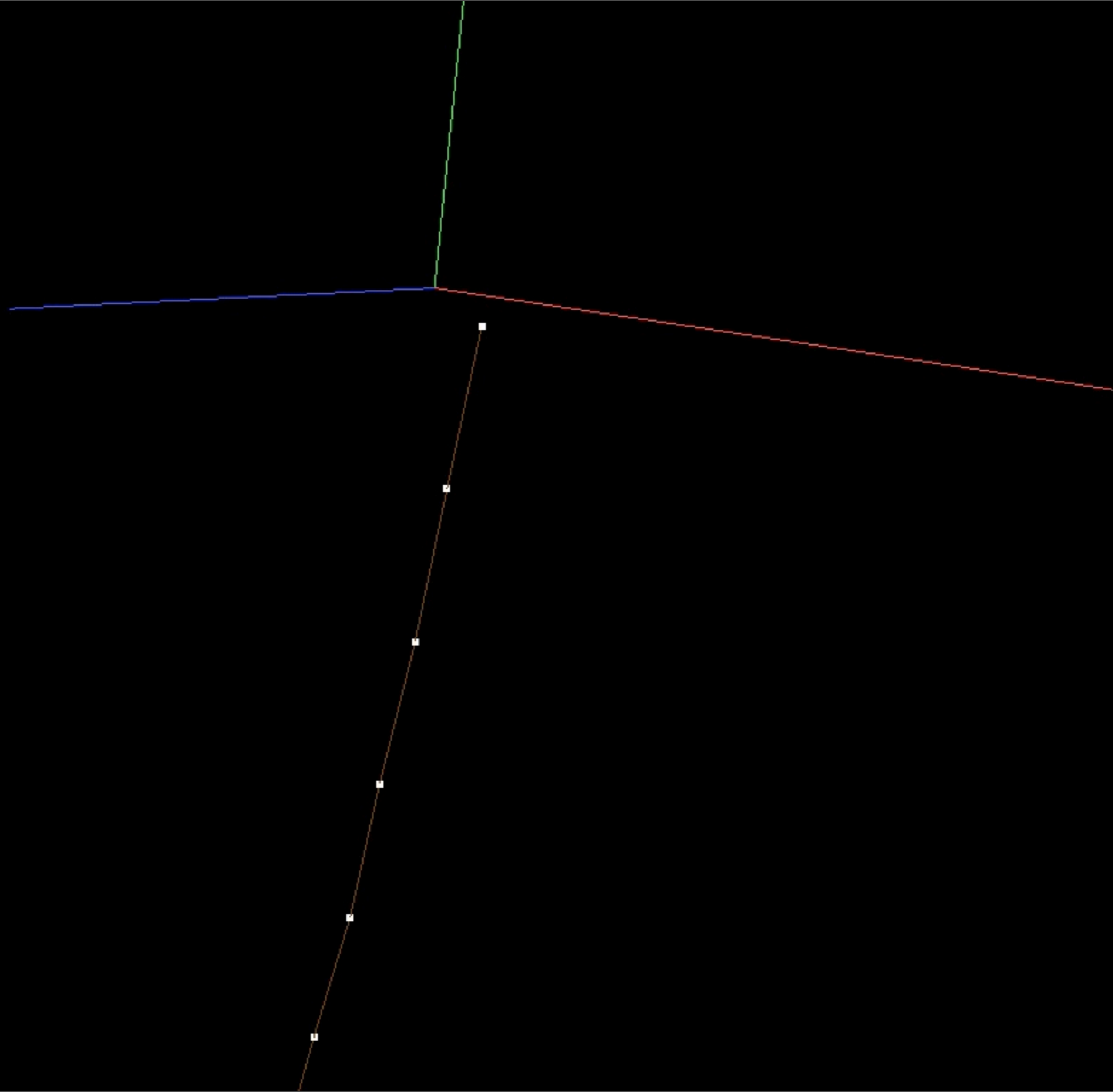
2fps
frame 0



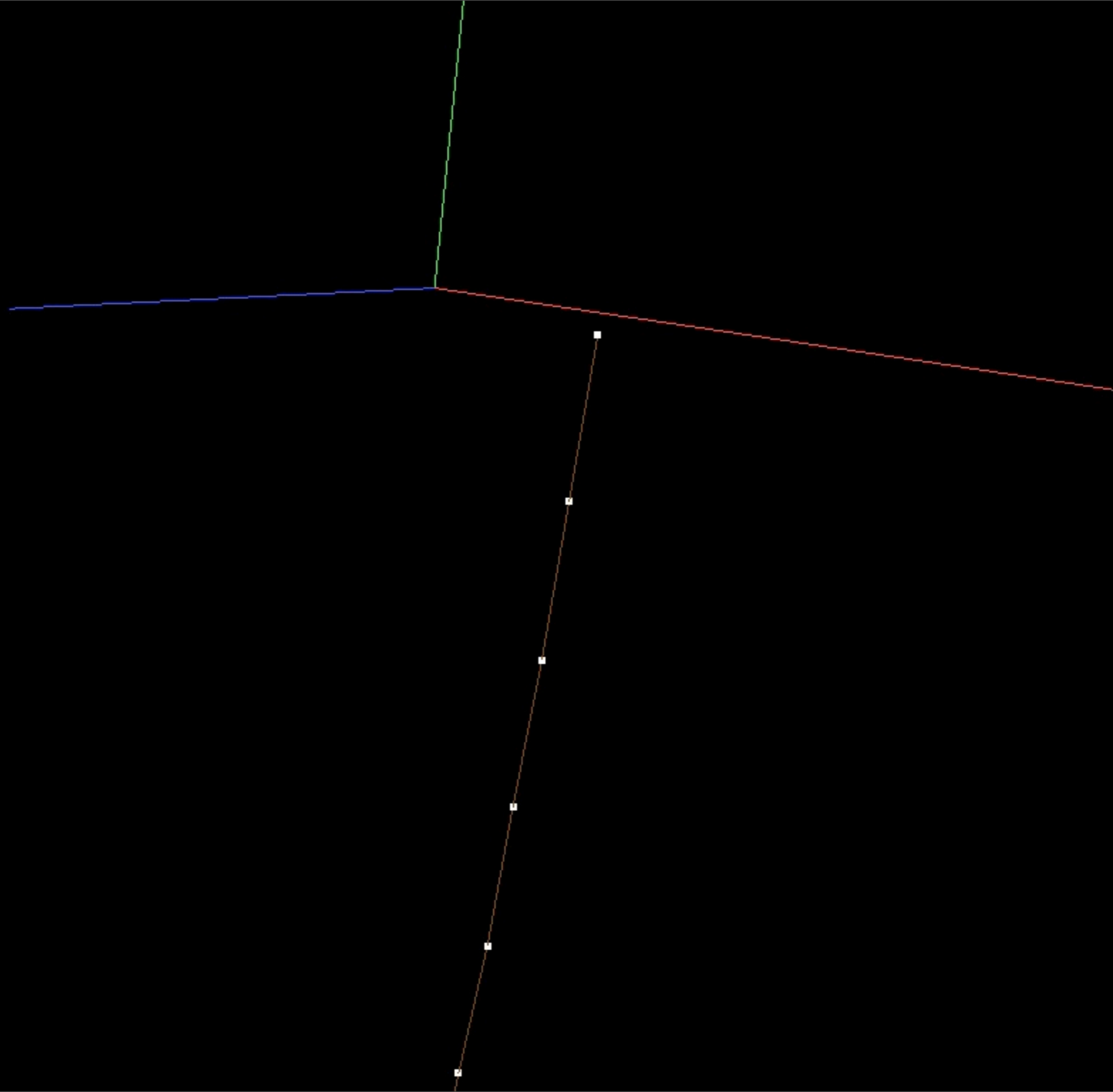
2fps
frame 5



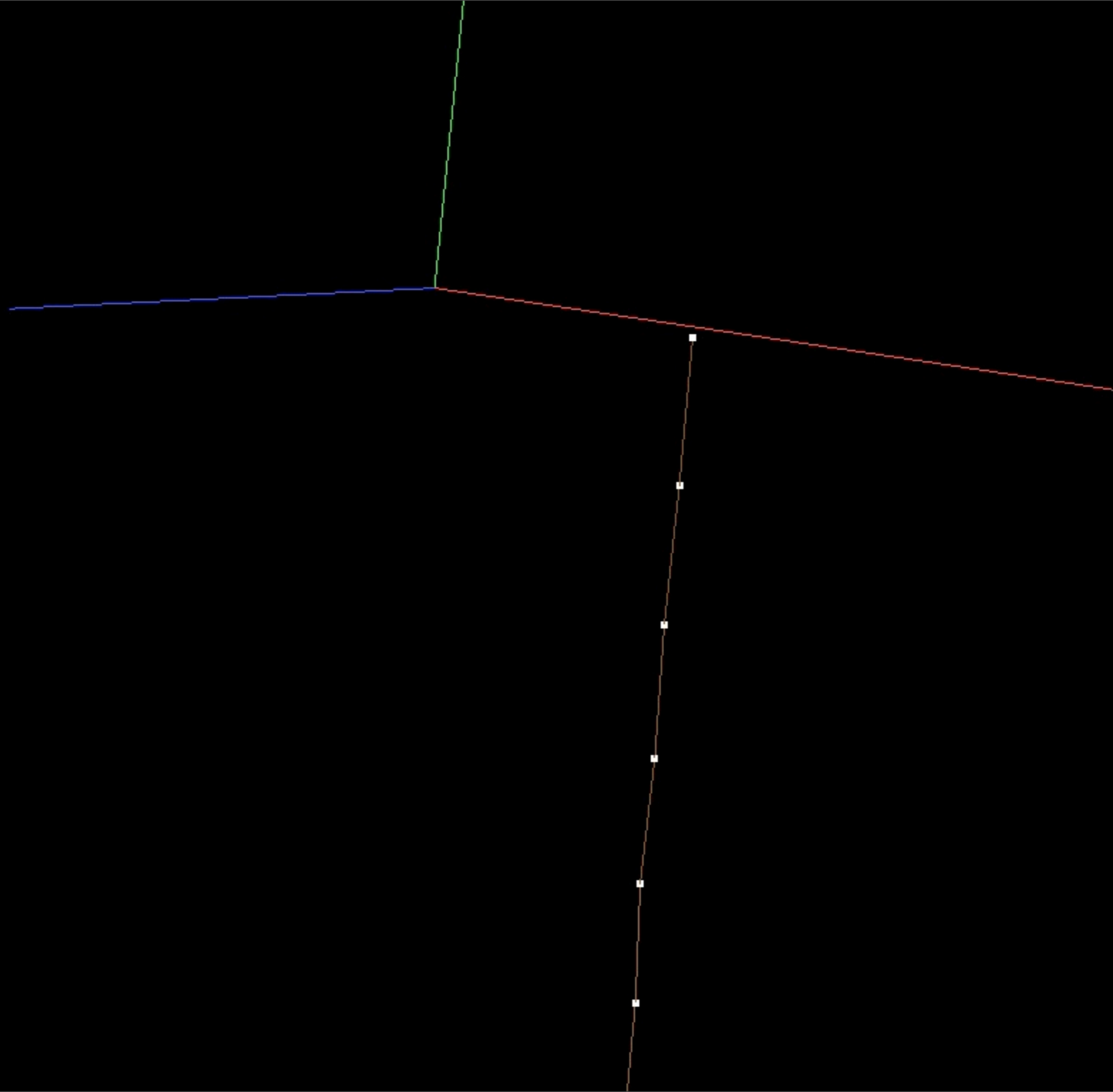
2fps
frame 10



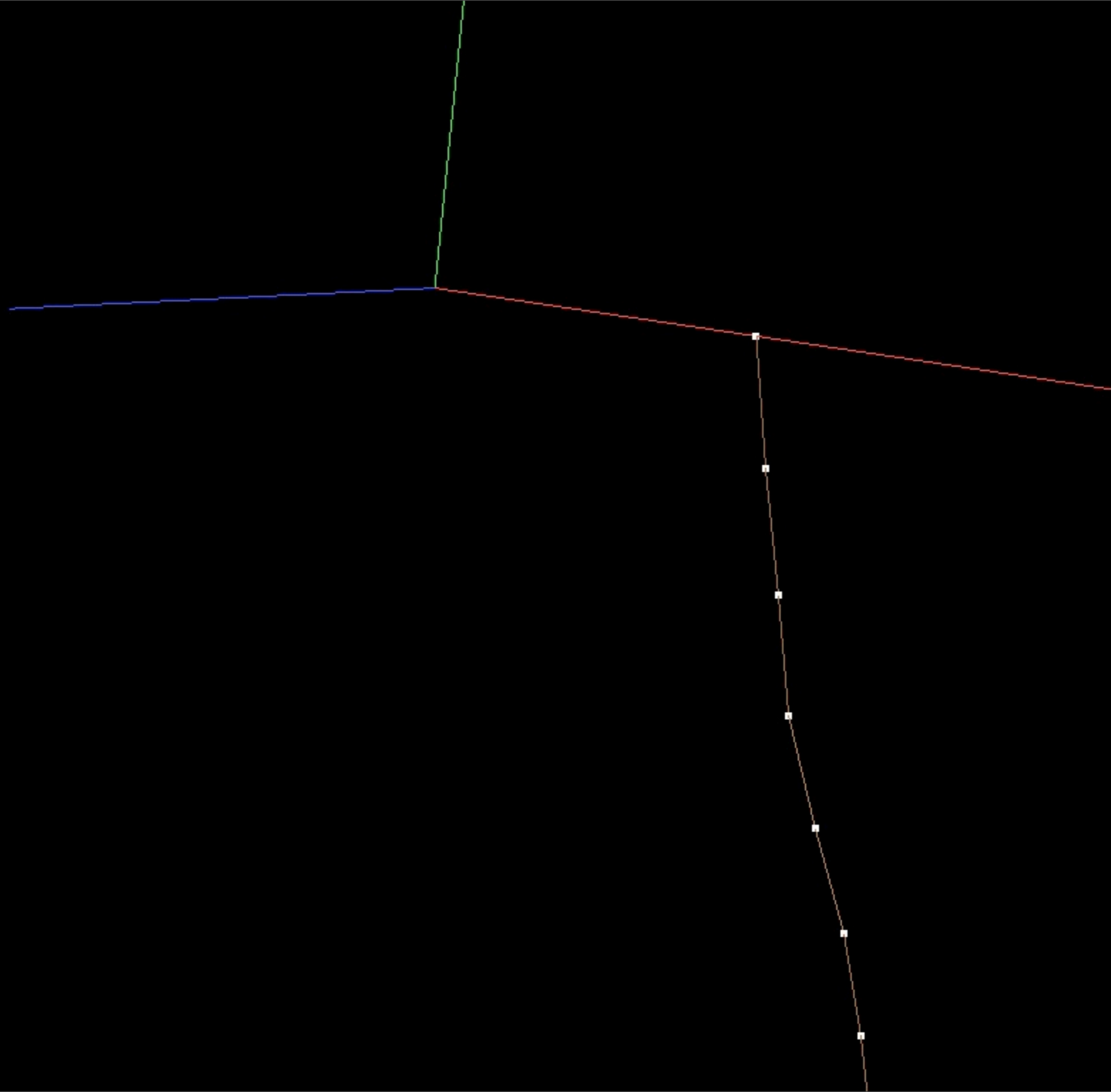
2fps
frame 15



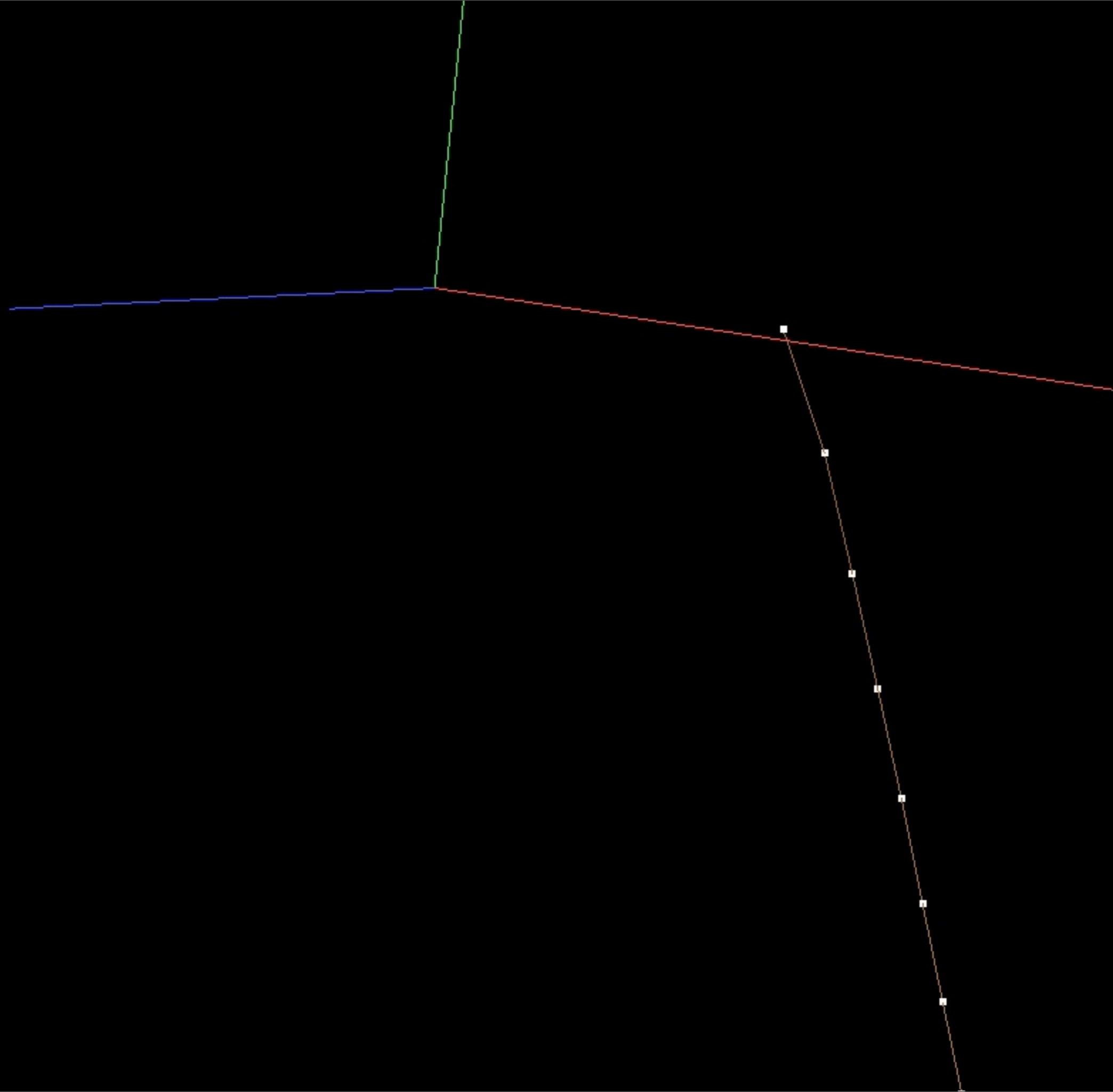
2fps
frame 20



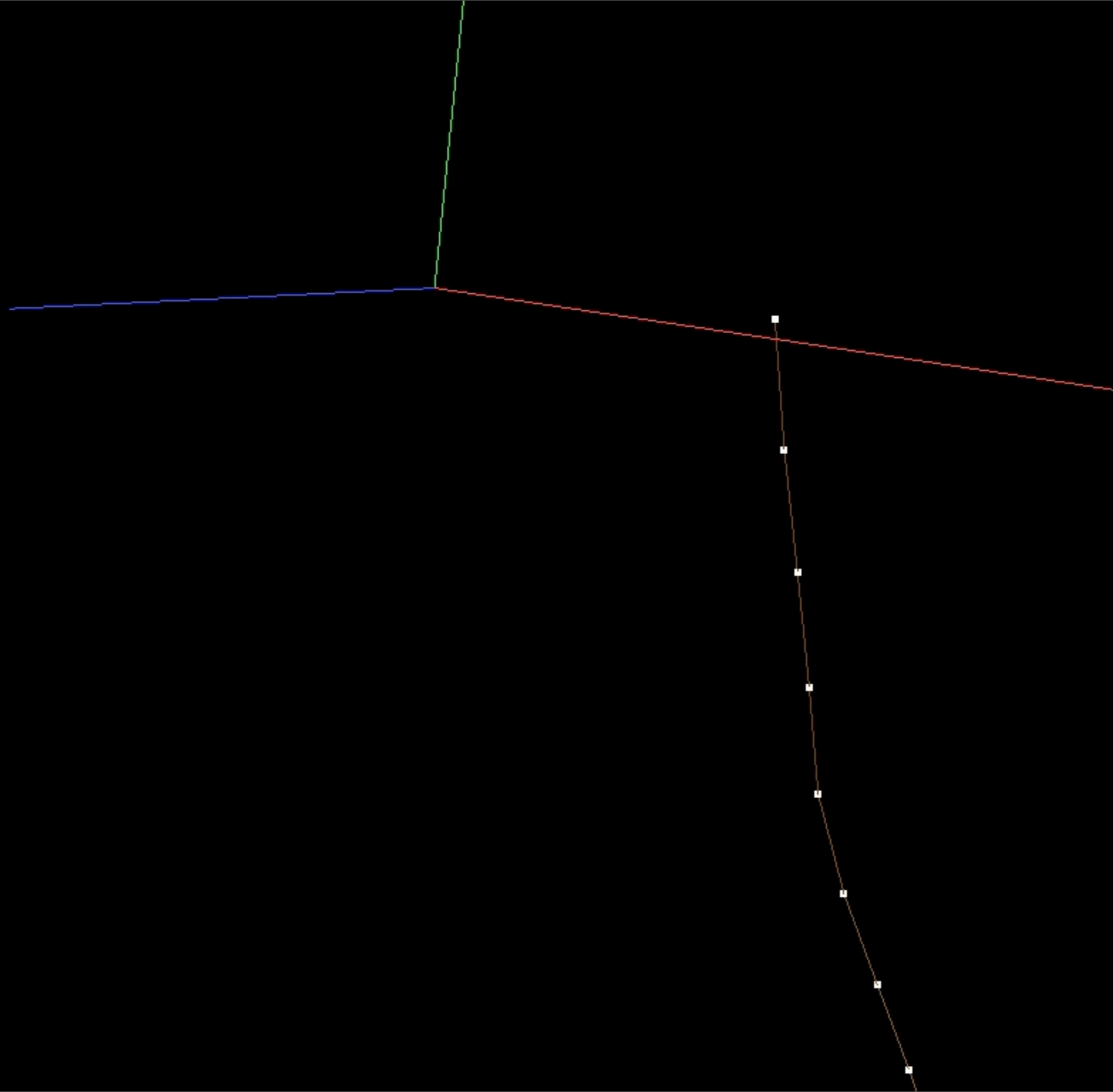
2fps
frame 25



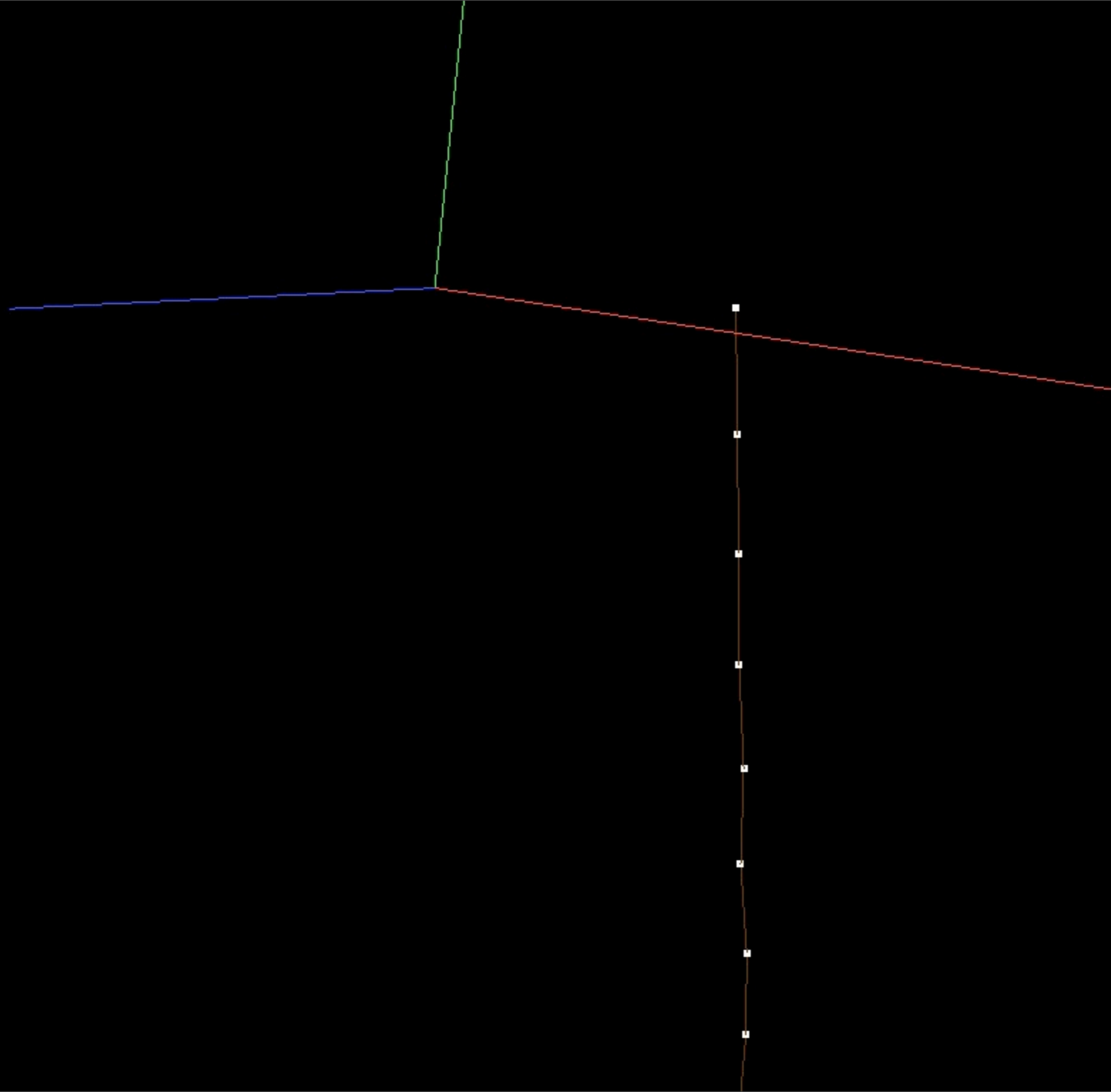
2fps
frame 30



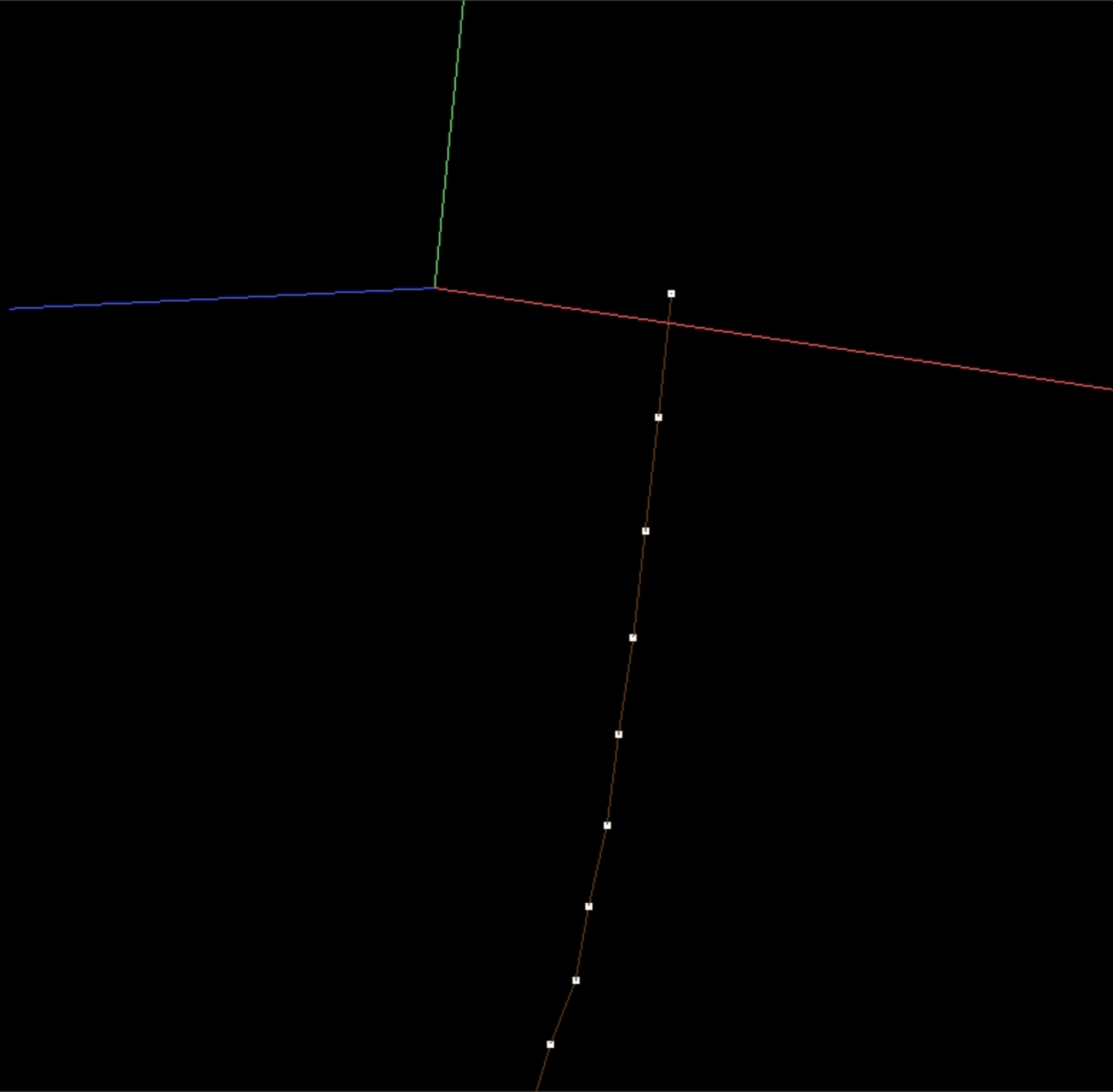
2fps
frame 35



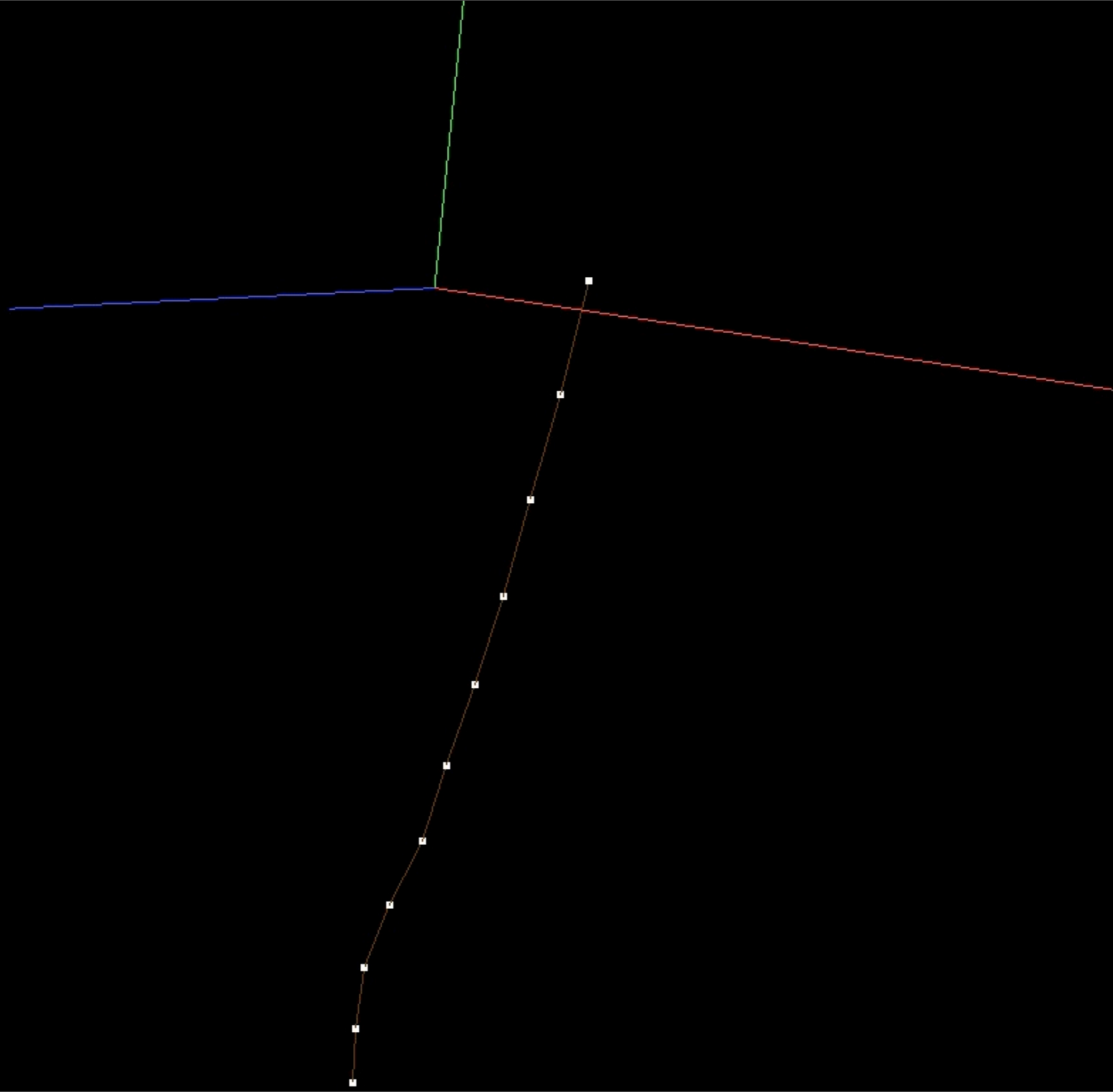
2fps
frame 40



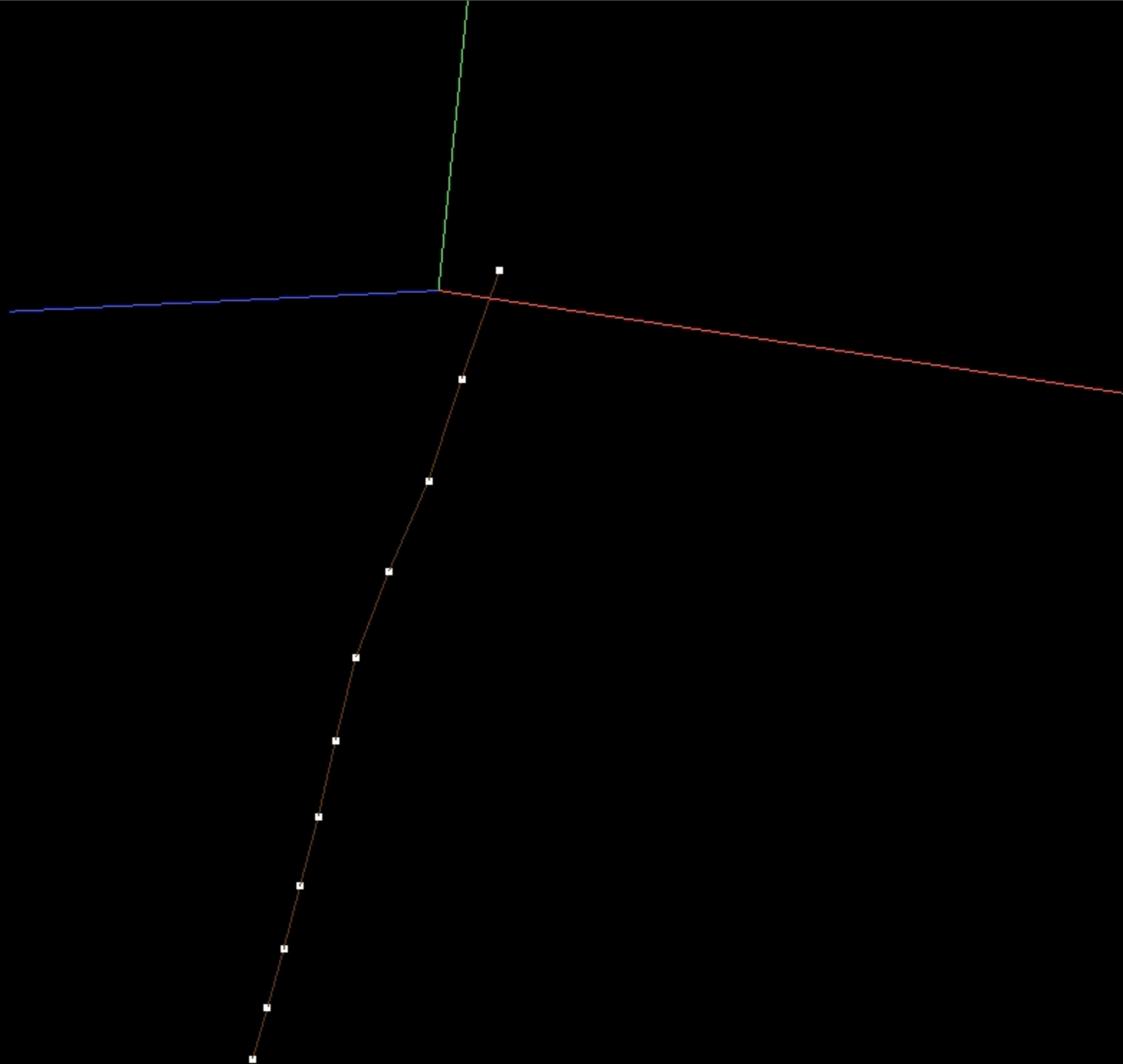
2fps
frame 45



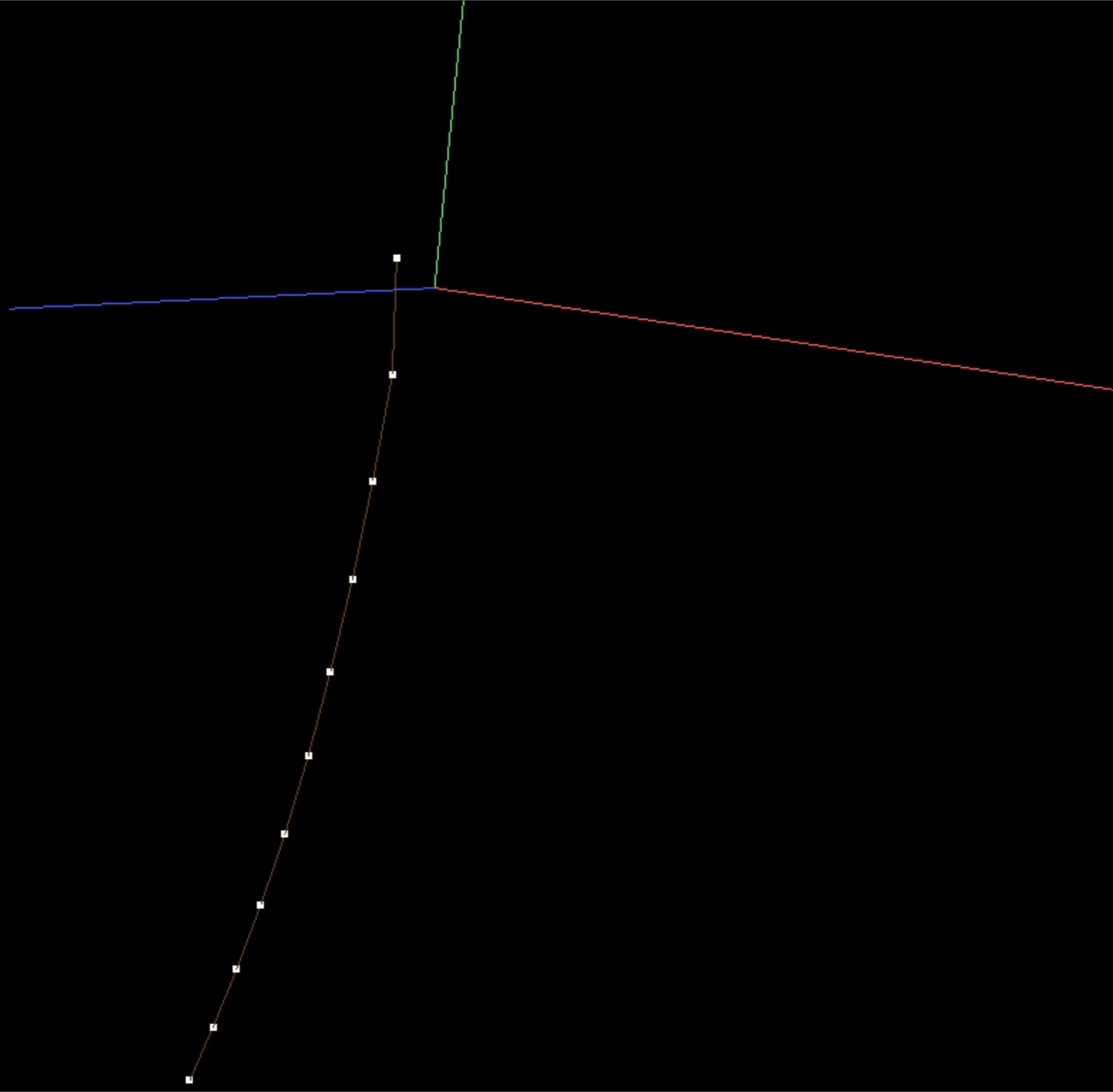
2fps
frame 50



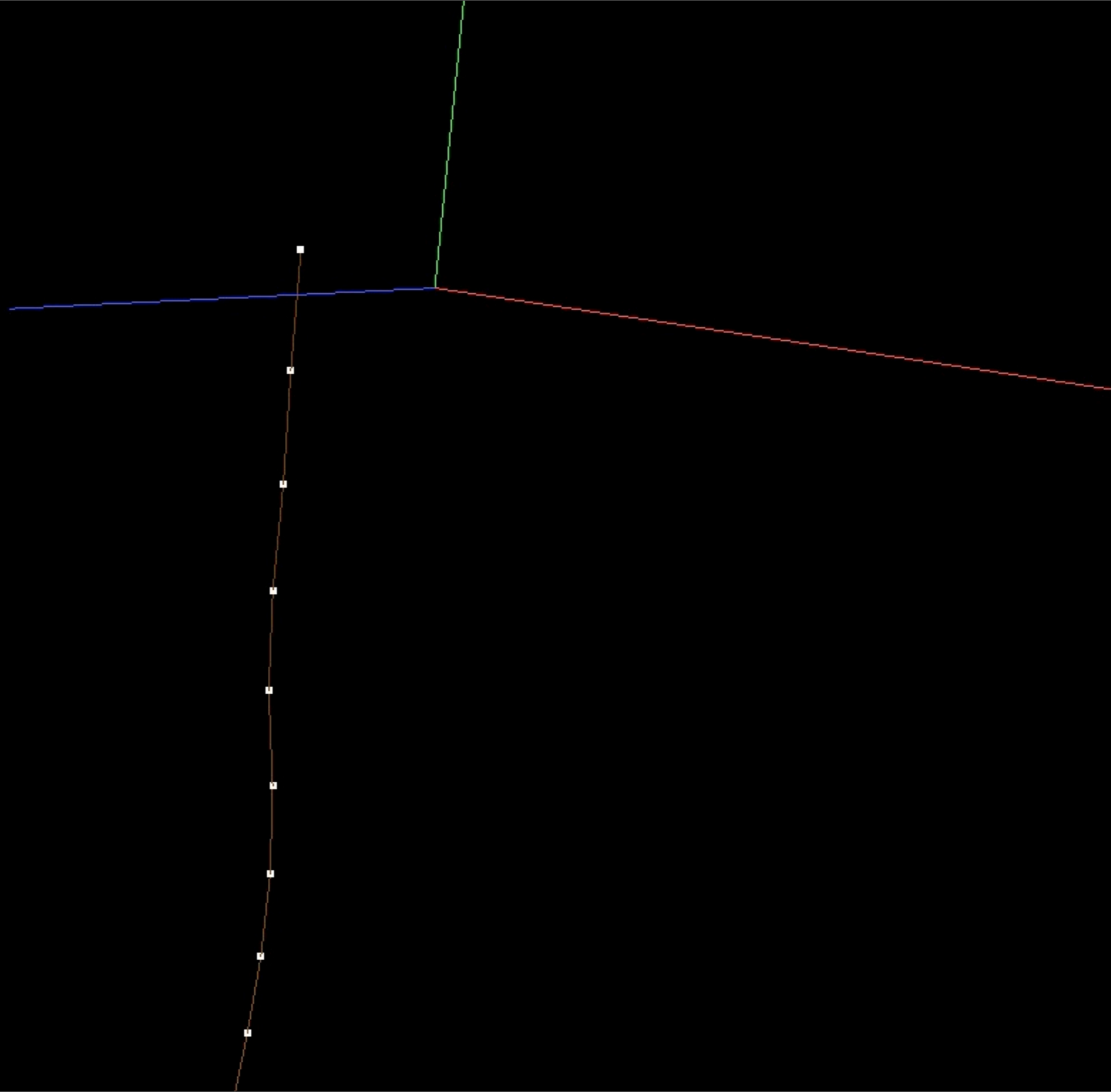
2fps
frame 55



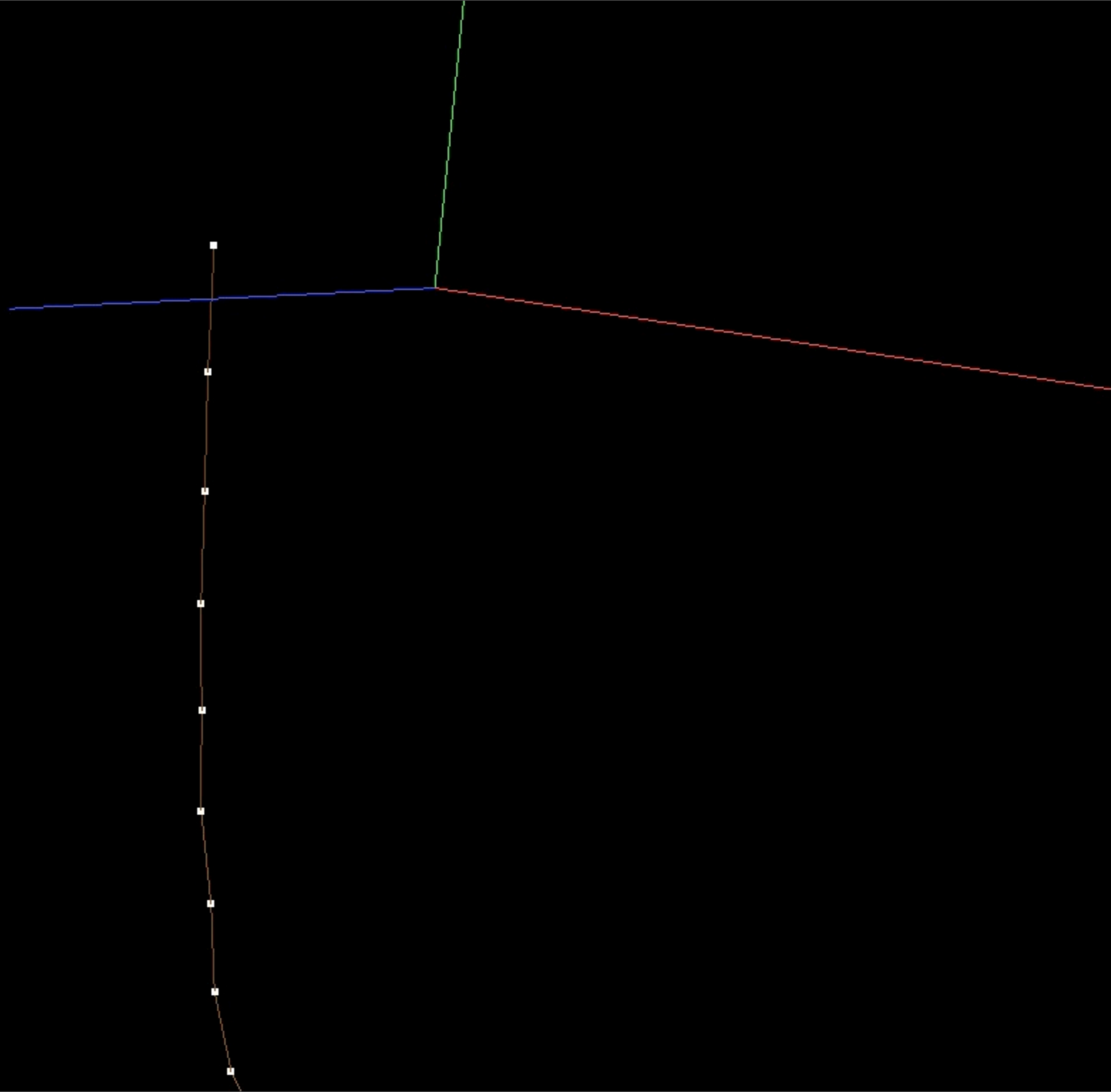
2fps
frame 60



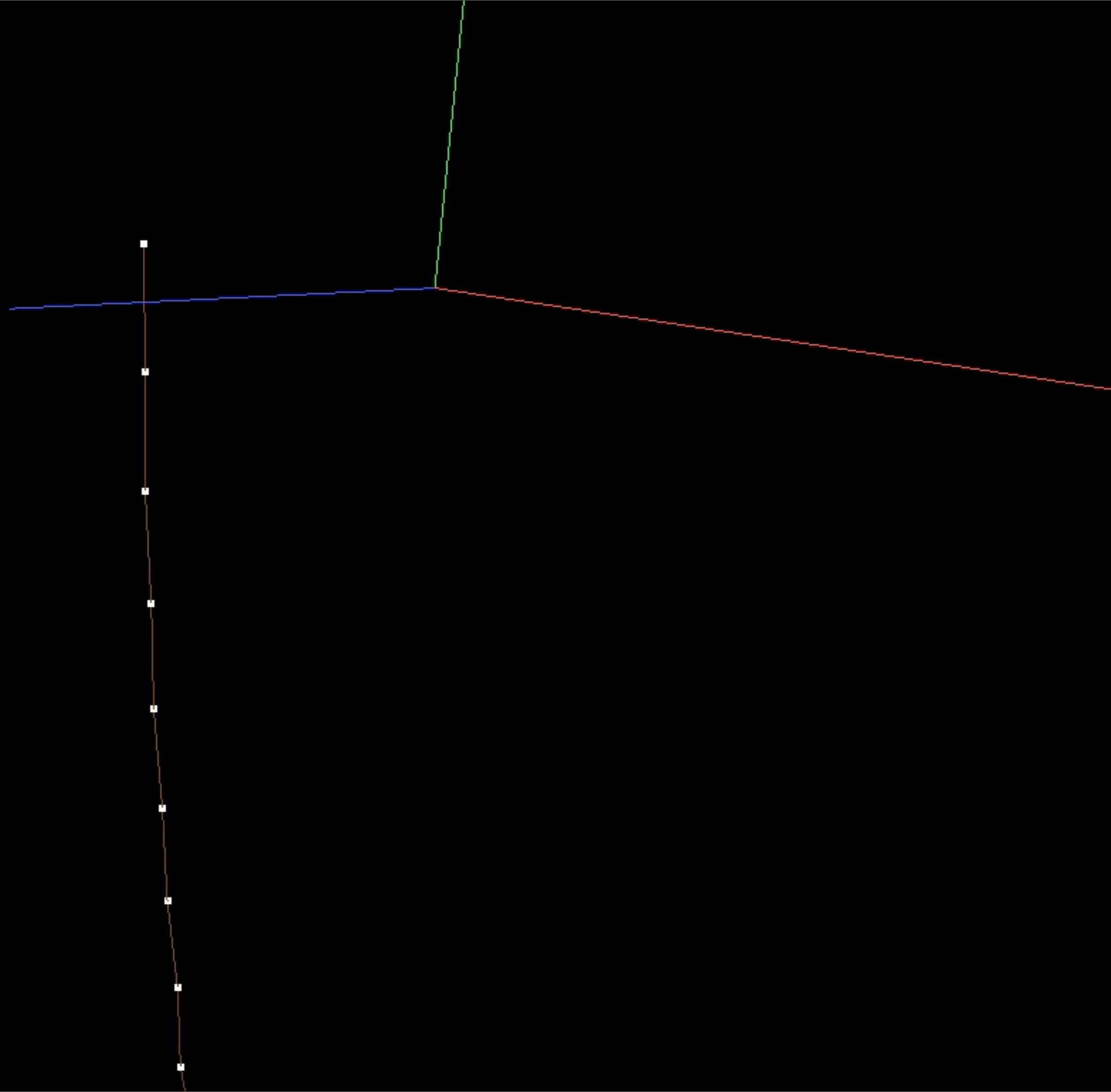
2fps
frame 65



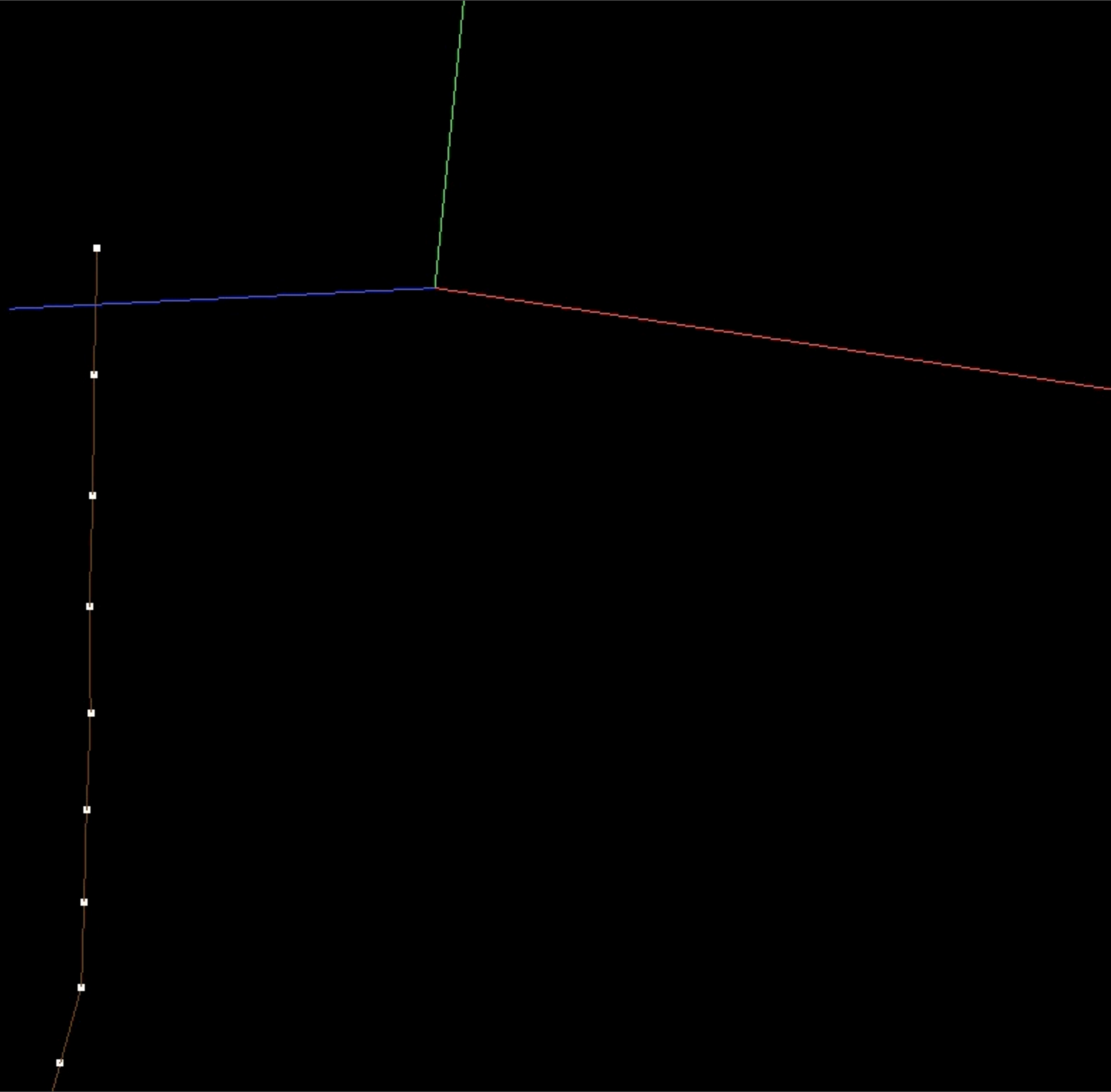
2fps
frame 70



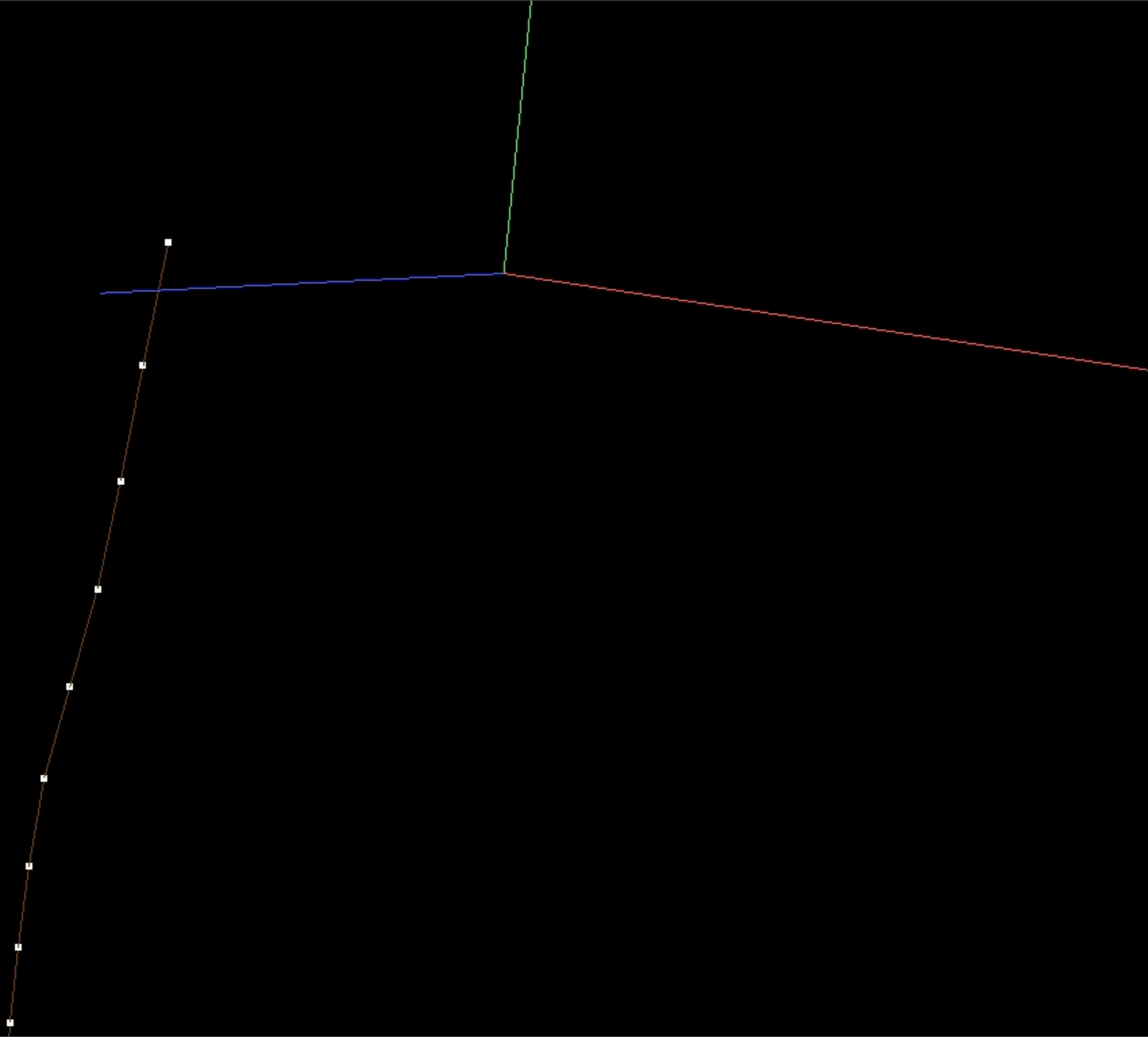
2fps
frame 75



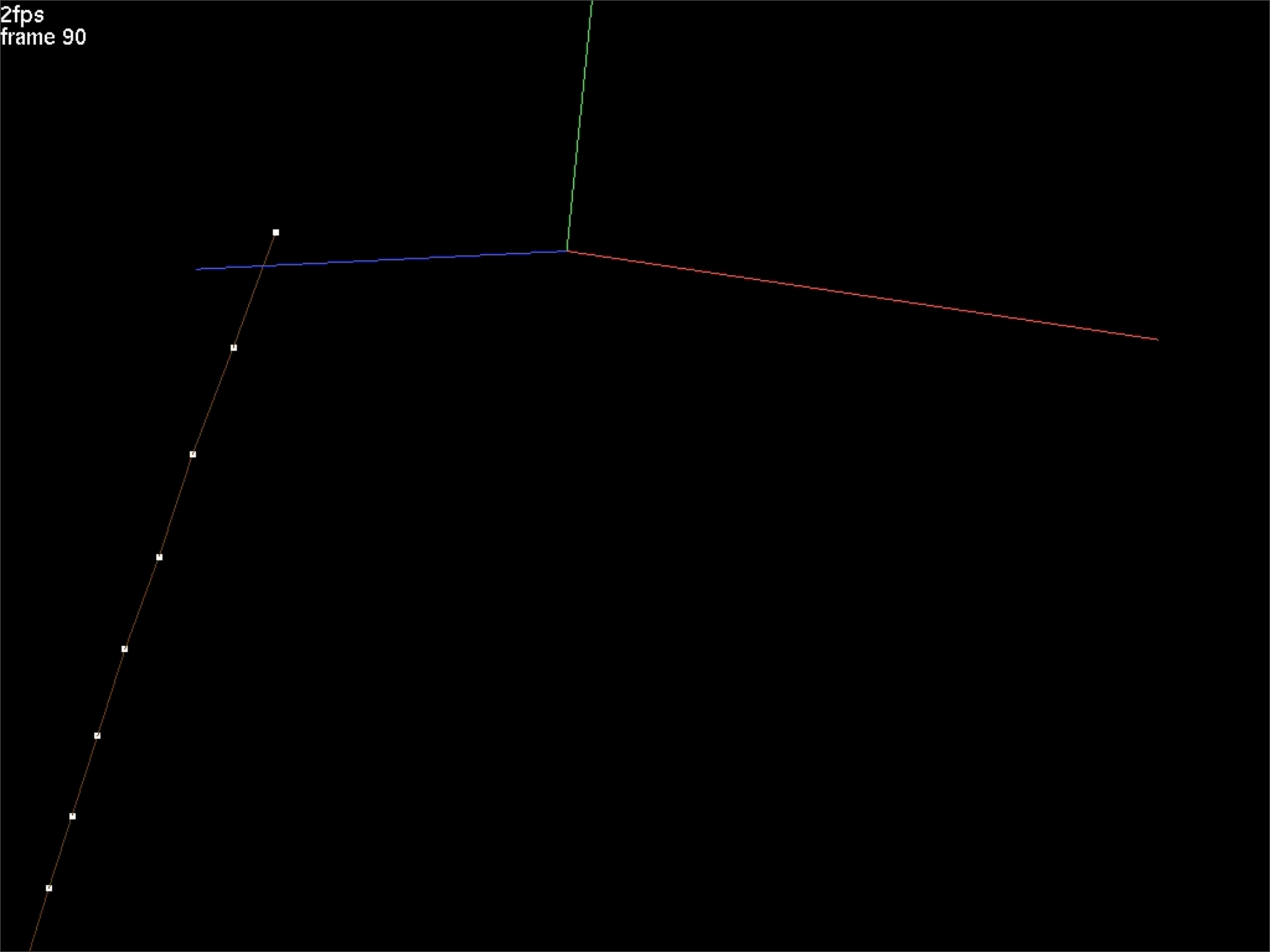
2fps
frame 80



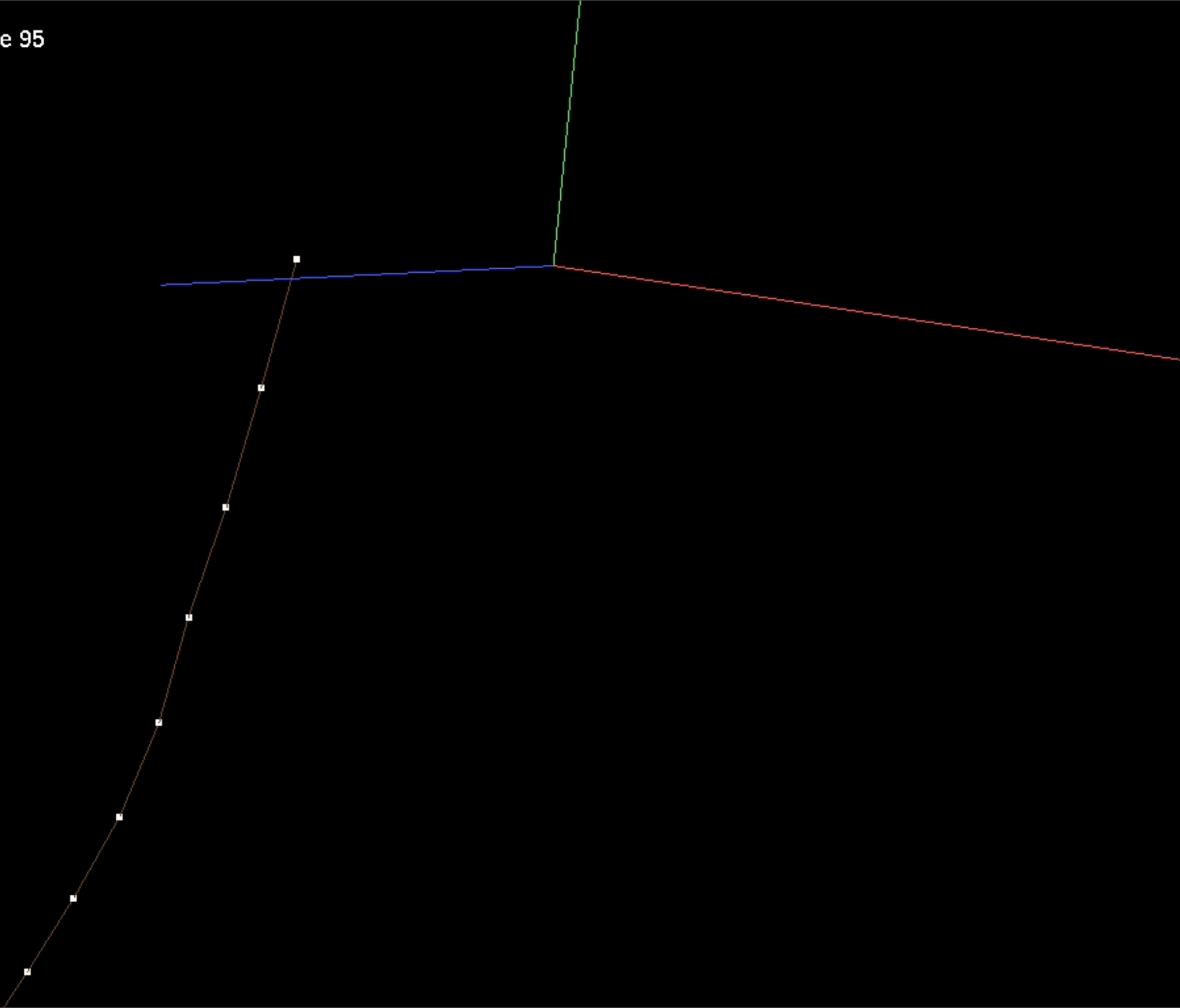
2fps
frame 85



2fps
frame 90



2fps
frame 95




```

[... ]

ARRAY<TV> force(n),dX(n),dV(n);

T dt_damping=mass*restlength/damping_coefficient;
T dt_elastic=damping_coefficient*restlength/youngs_modulus;
T CFL_number=0.5;
T dt_max=CFL_number*std::min(dt_damping,dt_elastic),dt;
T time=0.;

LOG::cout<<"dt_damping="<<dt_damping<<std::endl;
LOG::cout<<"dt_elastic="<<dt_elastic<<std::endl;
LOG::cout<<"Maximum dt="<<dt_max<<std::endl;

FILE_UTILITIES::Create_Directory("output/0");collection.Write(stream_type,"output",0,0,true);

for(int frame=1;frame<=number_of_frames;frame++){
    T frame_end_time=frame_time*(T)frame;

[... ]

        // Apply the Forward Euler method
        dX=dt*particles.V; // Compute position change
        dV=(dt/mass)*force; // Compute velocity change
        dX(1)=dV(1)=TV(); // First particle has externally prescribed motion; do not alter
        particles.X+=dX; // Update particle positions and velocities
        particles.V+=dV;

        FILE_UTILITIES::Create_Directory("output/"+STRING_UTILITIES::Value_To_String(frame));
        collection.Write(stream_type,"output",frame,0,true);
    }
}

LOG::Finish_Logging();
}

```

[...]

```
ARRAY<TV> force(n),dX(n),dV(n);
```

```
T dt_damping=mass*restlength/damping_coefficient;  
T dt_elastic=damping_coefficient*restlength/youngs_modulus;  
T CFL_number=0.5;  
T dt_max=CFL_number*std::min(dt_damping,dt_elastic),dt;  
T time=0.;
```

```
LOG::cout<<"dt_damping="<<dt_damping<<std::endl;  
LOG::cout<<"dt_elastic="<<dt_elastic<<std::endl;  
LOG::cout<<"Maximum dt="<<dt_max<<std::endl;
```

```
FILE_UTILITIES::Create_Directory("output/0");collection.Write(stream_type,"output",0,0,true);
```

```
for(int frame=1;frame<=number_of_frames;frame++){  
    T frame_end_time=frame_time*(T)frame;
```

[...]

```
    // Apply the Forward Euler method  
    dX=dt*particles.V; // Compute position change  
    dV=(dt/mass)*force; // Compute velocity change  
    dX(1)=dV(1)=TV(); // First particle has externally prescribed motion; do not alter  
    particles.X+=dX; // Update particle positions and velocities  
    particles.V+=dV;
```

```
    FILE_UTILITIES::Create_Directory("output/"+STRING_UTILITIES::Value_To_String(frame));  
    collection.Write(stream_type,"output",frame,0,true);
```

```
    }  
}
```

```
LOG::Finish_Logging();
```

```
}
```