Edinburgh Napier University

SET09119 Physics-Based Animation

Workbook

Sam Serrels

September 29, 2016

Contents

List of Figures

List of Algorithms

Chapter 1

Getting Started

1.1 Welcome

Welcome to Physics-Based Animation! This module and accompanying workbook will guide you through the process of simulating the vast mechanisms of real-word physics within software. Expect collisions, explosions, particles, and plenty of cubes and spheres rolling around everywhere.

The pre-requisite knowledge you require is:

- A Strong grasp of mathematical concepts such as trigonometry, algebra, and geometry. Linear algebra and matrix mathematics are crucial skills for this module.
- An adept or above level of skill with the C++ language.
- Knowledge of Computer Graphics technology and theory.
- Strong software debugging skills
- A willingness to spend time solving the problems presented in this workbook. Some of the exercises are challenging and require effort. There is no avoiding this. However, solving these problems will significantly aid your understanding.

1.2 Workbook Style

The workbook is task and lesson based, requiring you to solve problems to complete the code provided as a starting point for the lesson. This code will either be in the main application source file (`main.cpp`) for the lesson, or in one or more shader files (typically found in the `resources/shaders` folder). The sections you have to complete are highlighted in stars:

```
1 \\ *********************************
2 \\ You have to complete this problem
3 \\ *********************************
```

The problems faced generally build on previous lessons until you become familiar with the work involved.

1.3 Physics Framework

This module provides example code for you to build upon throughout the different topics. To make life easier, a small physics library has been included, this contains some useful helper functions and abstracts most of the graphics logic away from the main tutorial code. This physics library is built on top of the Napier graphics framework, which you should be familiar with. The purpose of this module is to teach physics, not graphics, however; the two are closely linked so you may need to dive down into the libraries to write/modify some graphics code.

1.3.1 Libraries

Other than the libraries used by the graphics framework (GLEW,GLFW,ASSIMP...) the physics code makes use of the following libraries:

GLM the GL Mathematics library. GLM is also cross-platform and a header only library (it has a number of different headers we include for different purposes). GLM provides the mathematical types and functions required to work with OpenGL (which provides no basic mathematical types).

IMGUI , a small graphical user interface library for C++, is particularly suited to integration in realtime 3D applications. It favours simplicity and productivity and is easy to implement and use.

1.4 Getting the Code - Clone The Repository from GitHub

Everything you need to get started (including this workbook) is hosted in a git Repository. You will need to clone your own copy to work with.

```
1 git clone https://github.com/NapierUniversity/enu_pba.git
```

1.5 Configuring the code - CMake

1.5.1 Intro to CMake

The physics project makes use of the CMake build system, this allows for cross-platform compatibility, dependency acquisition and general purpose code housekeeping. If you are new to CMake, here is a simple explanation of its purpose.

CMake uses a configuration script to generate Make Files (or IDE Project files) for your project. Instead of shipping a makefile which would have to be updated, and would be specific to one platform, only the project code and a CmakeLists.txt config script is needed. The same goes for Visual studio Solution files, which are large, difficult to version, and can only be used by Visual studio.

Instead of maintaining a visual studio sln file, CMake will generate one for you. CMake does not actually *build* your code, it just *configures* your build environment for you so you can build the software.

CMake is also used to pull down the additional libraries we will need for this project. This means that you are getting a fresh and up-to date copy of all the libraries, and you will be compiling them specifically for your system. Say goodbye to linker and VCRuntime errors.

1.5.2 Running CMake

To keep the source directory (the directory pulled from Git) clean, we tell CMake to generate our build files in a separate directory, This is called an Out of Source Build. We can do this all from the command line easily:

```
1 mkdir physics_build
2 cd physics_build
3 cmake -G "Visual Studio 14 2015 Win64" ../enu_pba/
```

CMake Has a GUI interface also, this is useful if you are configuring new software and need to see and set options. This project has all the settings set to sensible defaults so you should not need to touch anything other than the configure and generate buttons. The steps are Shown in Figure 1.1.

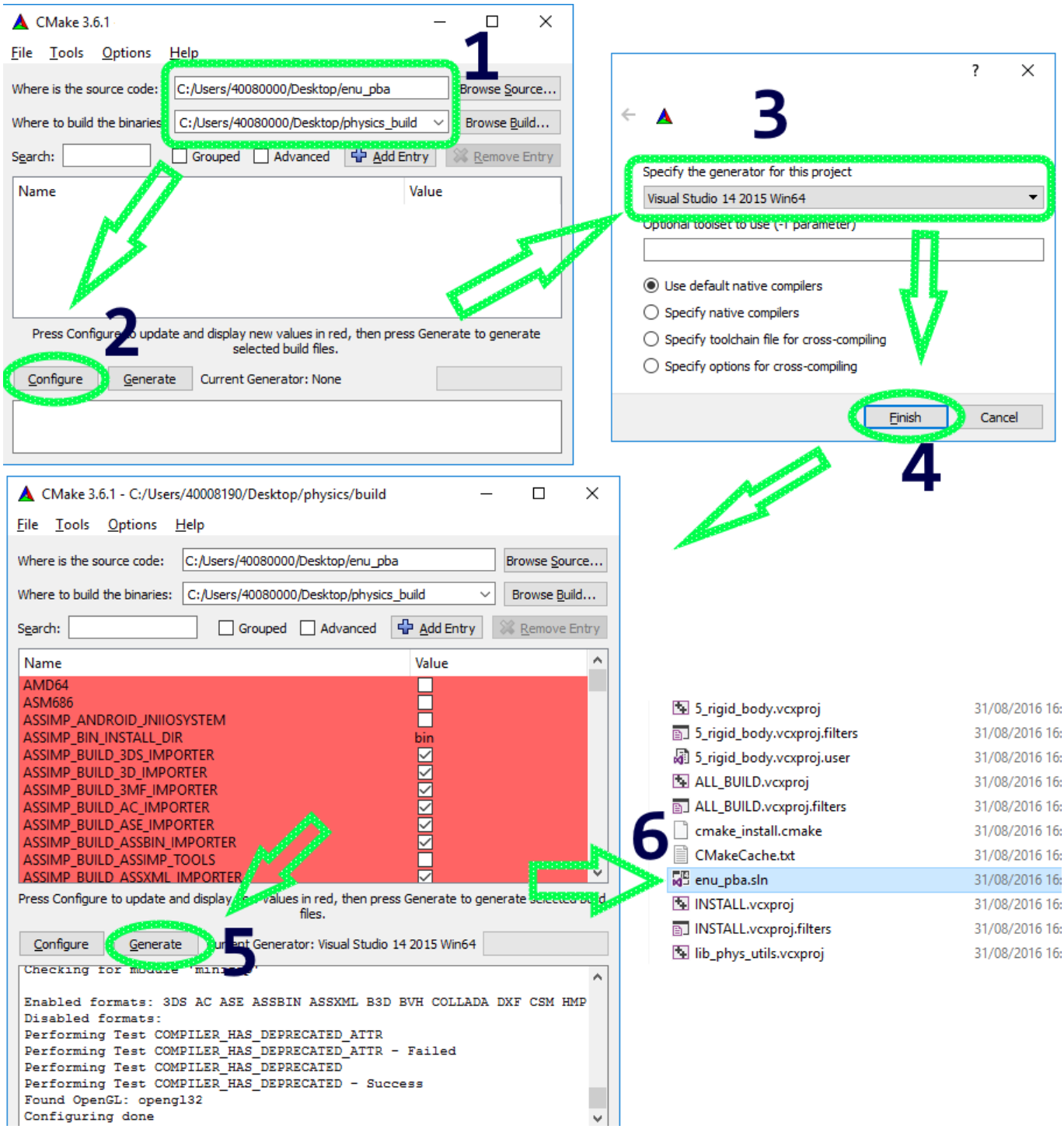

Figure 1.1: **CMake GUI Build Steps**

1.6 Building the code - Visual Studio

Once CMake has created the Solution file and you have opened it, you should see the following projects:

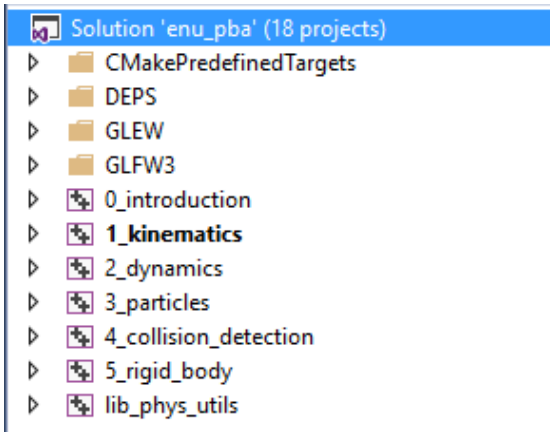

Figure 1.2: **Visual Studio projects**

The projects within the sub-folders are the libraries which we will use, so you don't need to pay attention to them. The CMakePredefinedTargets contains some CMake functions, specifically useful is the ZERO_CHECK project, this will trigger a CMake configure and rebuild without having to leave VS. This is useful if anything in the source directory has changed.

1.7 Running the Code

You should run this application to see the output. You will get a window as shown in Figure 1.3.

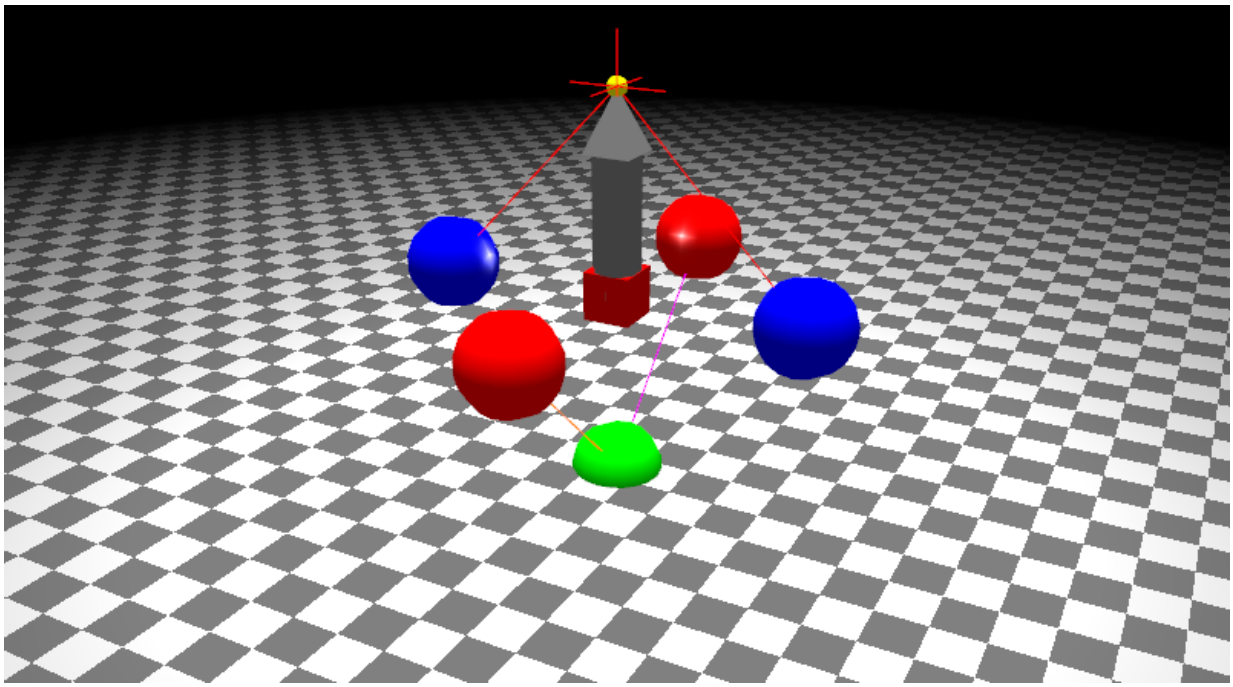

Figure 1.3: **Output from Basic Application**

Chapter 2

Inverse Kinematics

2.1 Introduction

The topic of this practical is the implementation of a real-time inverse kinematic simulation (i.e., a linked chain of rigid body objects). The user should be able to interact with the simulation while it is running (e.g., through the mouse or keyboard) to control the inverse kinematic target end-effector. The inverse kinematic chain can be implemented using either the cyclic coordinate descent (CCD) algorithm or the Jacobian pseudo-inverse matrix method.

Tasks

1. Visually display interconnected chain of 3D rigid bodies (i.e., base and end-effector)
2. Implement an uncomplicated single-joint IK system using an analytical method (i.e., axis-angle) which follows the mouse cursor around the screen (Section 2)
3. Implement a linked-chain of interconnected limbs using an iterative inverse kinematic technique (e.g., CCD IK with details given in Section 3)
4. User input (e.g., mouse or keyboard) to control and move the end-effect target
5. Areas to explore, include, altering the number of links (e.g., 10 to 1000). Adding physical constraints (i.e., limit the joint angles +/- specified amount). Modify the code to support multiple end-effectors (e.g., tree-like structure)

2.2 Overview

Principles and Concepts The application of kinematic algorithms for animation is used to control articulated postures based on a simple definition of joint angles and limb lengths. These structures may take practically any form from a humanoid bipedal character to just about anything that can be imagined using a hierarchy.

Hierarchy Structures described using a hierarchical form are defined using a parent-child system similar to that of tree structure. In the case of computer characters, each rigid limb of the structure is a child node in the tree whose parent node provides a reference point from which it is described. The parents are themselves child nodes of limbs above in the hierarchy and this recursive relationship

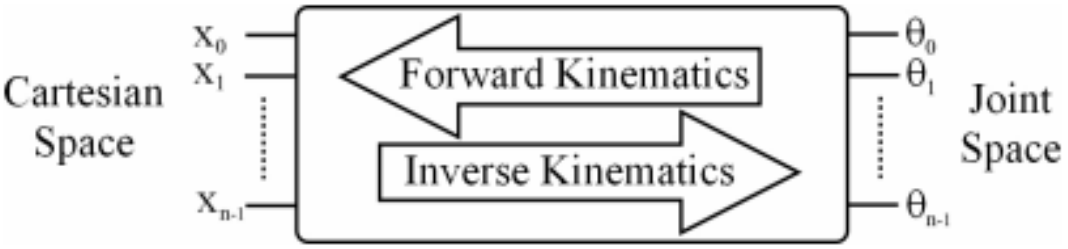

Figure 2.1: **Forward & Inverse Kinematics**Illustrating the relationship between forward and inverse kinematics parameters

continues up to a root node (i.e., the bottom of the tree). The parent of the root node is effectively taken as the global frame of reference and defined as such.

End-Effectors At the top end of the tree are leaf nodes which are children that have no descendants of their own. An end-effector in terms of kinematic chains is any node within the hierarchy that an animator wishes to directly position, for example, to interact with the environment. End-effectors are commonly the leaf nodes of an articulated structure, such as, the feet and hands of an animated character since they generally interact with the world.

Graphics Inverse kinematics is a programmatic solution for controlling the animation of rigid models. When artists generate an animated model for a virtual environment, such as a game, the animations created by the artists wont necessarily work for every position in the world. For example, if an artist were to create an animation of a character pressing a button, the animation would only work so long as the button was always at the same position relative to the model. If the button were to be moved up or down the pre-made animation would have no way to account for that. With inverse kinematics, we can reconfigure the interconnected set of links (e.g., animate the model) so the end-effector (e.g., a hand) can be placed anywhere.

Inverse Kinematic (IK) Methods

1. Analytical (e.g., Geometric Analysis)
2. Heuristic (e.g., CCD)
3. Algebraic Solver (e.g., Jacobian & Gradient-Based Search)

2.3 Analytical Inverse Kinematics

The geometric/analytical algorithms tend to be very quick because they reduce the IK problem to a mathematical equation that need only be evaluated in a single step to produce a result (e.g., see Figure 2.2). The limitations of this class of solver becomes apparent in the case of large chains. In such cases, the task of reducing the problem to a single-step mathematical equation is impractical. Therefore geometric/analytical techniques tend to be less useful in the field of character animation.

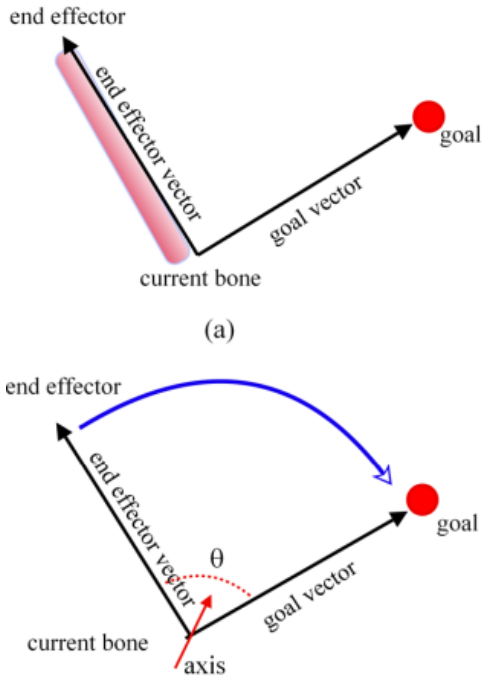

Figure 2.2: **Analytical Inverse Kinematic Example**Single limb and joint angle (solved using the dot & cross product)

2.4 Cyclic Coordinate Descent (CCD)

IK solvers that are based on CCD use an iterative approach that takes multiple steps towards a solution. CCD works by analysing each joint one-by-one in a progressive refinement philosophy. Starts with the last joint in the chain (e.g., a hand for a character) and tries to rotate it to orientate it toward the target. The steps that the method takes are formed heuristically, therefore each step can be performed relatively quickly. An example of a possible heuristic would be to minimise the angle between pairs of vectors created when projecting lines through the current node and end-effector and current node and desired location. However, because the iterative step is heuristically driven, accuracy is normally the price paid for speed. Another issue with this technique is that one joint angle is updated at a time as opposed to the complete hierarchical structure. This has the undesirable and unrealistic result of earlier joints moving much more than later limbs in the IK chain.

Implementation The Cyclic Coordinate Descent (CCD) algorithm is an iterative IK solution. The basic idea of the CCD algorithm is to loop over each bone in the IK chain and rotate it such that the end effector (which is typically the last bone in the chain) will move as close as it can to the final position

while *(distance from end effector to target* $\geq$ *threshold)* $\wedge$ *(numloops* $<$ *max)* **do**
    Take current bone;
    Build vector V1 from bone pivot to end effector;
    Build vector V2 from bone pivot to target;
    Get the angle between V1 and V2;
    Get the rotation direction;
    Apply a differential rotation to the current bone;
    if *current bone is the last bone in the chain* **then**
        restart at first bone;
    else
        current bone is the next bone in chain;
    end
end

Algorithm 1: Algorithm for the CCD system

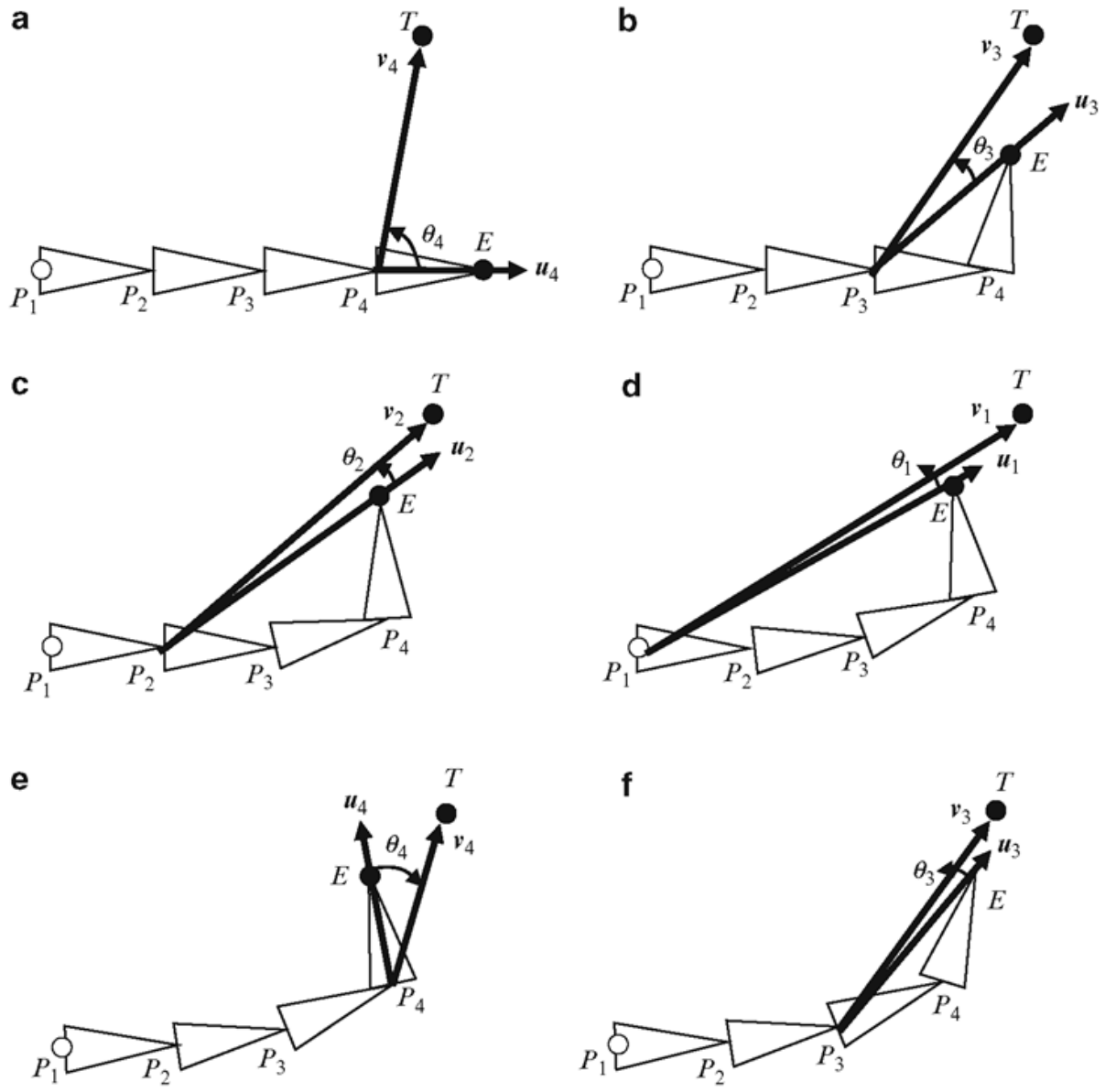

Figure 2.3: **Sequence of rotations performed by CCD algorithm on a 4-link joint chain**[3]

Figure 2.3 shows the principle of the CCD technique. (a) Shows The initial state

(b) The end effector joint P_4 has been moved to bring the vector U_4 as close as possible to V_4, the next vector for the next joint P_3 is then calculated.

(c) The joint P_3 has been moved to bring the vector U_3 as close as possible to V_3, this brings the end effector E close to the target T

(d) The Process repeats for the last joint

(e..f) The process restarts from the end joint P_4

2.5 Iterative Jacobian Pseudo Inverse

Due to their scalability, numerical techniques often form part of an inverse kinematics solver. However, because of their iterative nature, such methods can be slow. So far research into the field of kinematics has failed to find a general nonnumerical solution to the problem. Many researchers have proposed hybrid techniques yet these still rely on a numerical aspect. It is therefore important to find ways of using numerical techniques as efficiently as possible. In this paper we take a look at the Jacobian-based IK solver and techniques that allow this method to be used as an efficient real-time IK solver.

The end effector velocities are related to the joint velocities of the robot through the Jacobian matrix as follows:

$$\dot{e} = J\dot{\theta}$$

Where J is the Jacobian representing the partial derivatives for the change in end-effectors locations with change in joint angles. Typically, the system of joints consists of a non-square Jacobian, hence, we need to perform the pseudo inverse operation to yield the joint velocities and solve for the angular displacement:

$$J^{+} = J^{T}\left(JJ^{T}\right)^{-1}$$

$$\dot{\theta} = J^{+}\dot{e}$$

Where we are able to connect $\dot{\theta}$ for change in joint angles with change in end-effector location e. This method sets the angle values to the pseudo inverse of the Jacobian. It tries to find a matrix which effectively inverts a non-square matrix. It has singularity issues which tend to the fact that certain directions are not reachable. The problem is that the method first loops through all angles and then needs to compute and store the Jacobian, perform a pseudo inversion, calculate the changes in the angle, and last apply the changes.

while *distance from end effector to target* $\geq$ *threshold* **do**
  Compute $J(e, \theta)$ for the current pose;
  Compute J^{-1} i.e., invert the Jacobian matrix;
  $\Delta e = \beta(t - e)$ - pick approximate step to take;
  $\Delta\theta = J^{-1}\Delta e$ - compute change in joint DOFs;
  $\theta_{current} = \theta_{previous} + \Delta\Theta e$ - apply change to DOFsFs;
end

Algorithm 2: Jacobian System

2.5.1 Calculating the Jacobian Matrix

The Jacobian J is a matrix that represents the change in joint angles$\Delta\theta$ to the displacement of end-effectors Δe. Each frame we calculate the Jacobian matrix from the current angles and end-effectors. We assume a right-handed coordinate system. To illustrate how we calculate the Jacobian for an articulated system, we consider the simple example shown in Figure 2.4

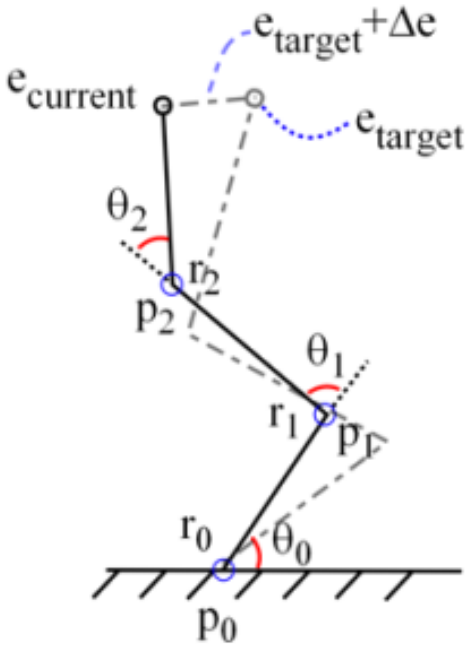

Figure 2.4: **2D 1DOF Jacobian IK Example** - Relationship between multiple joint angles and end-effectors.

Figure 2.4 demonstrates how we decompose the problem and represent it as a matrix for a sole linked chain with a single three degree of freedom (DOF) end-effector. We then extend this method to multiple linked-chains with multiple end-effectors (each with six DOF) to represent a structured hierarchy

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \dots \\ \theta_n \end{bmatrix} \quad e = \begin{bmatrix} e_{current_x} \\ e_{current_y} \\ e_{current_z} \end{bmatrix}$$

where θ is the rotation of joint i relative to joint i 1, and e for the end-effectors global position. The angles for each joint and the error for each end-effector are represented by matrices. From these matrices, we can determine that the end-effectors, and the joint angles are related. This leads to the forward kinematics definition, defined as:

$$e = f(\theta)$$

We can differentiate the kinematic equation for the relationship between end-effectors and angles. This relationship between change in angles and change in end-effectors location is represented by the Jacobian matrix.

$$\dot{e} = J(\dot{\theta})$$

The Jacobian J is the partial derivatives for the change in end-effectors locations by change in joint angles.

$$J = \frac{\delta e}{\delta \theta}$$

If we can re-arrange the kinematic problem:

$$\theta = f^{-1}(e)$$

We can conclude a similar relationship for the Jacobian:

$$\dot{\theta} = f^{-1}(\dot{e})$$

For small changes, we can approximate the differentials by their equivalent deltas:

$$\Delta e = e_{target} - e_{current}$$

For these small changes, we can then use the Jacobian to represent an approximate relationship between the changes of the end-effectors with the changes of the joint angles.

$$\Delta\theta = J^{-1}\Delta e$$

We can substitute the result back in:

$$\theta_{current} = \theta_{previous} + \Delta\theta$$

The practical method of calculating J in code is used:

$$\frac{\delta e}{\delta \theta_i} = r_j \times (e_{target} - p_j)$$

Where r_j is the axis of rotation for link j, e_{target} is the end-effectors target position, p_j is end position of link j

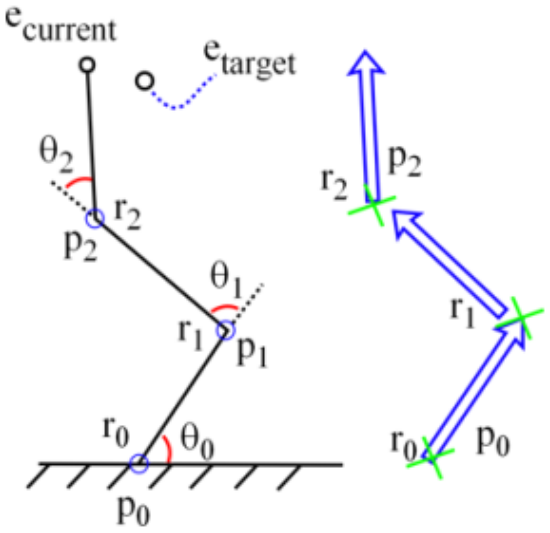

Figure 2.5: **Jacobian**- Iteratively calculating the Jacobian on a frame by frame basis.

For example, calculating the Jacobian for Figure 2.5 gives:

$$J = \begin{bmatrix} \frac{\delta e}{\delta \theta_0} \\ \frac{\delta e}{\delta \theta_1} \\ \frac{\delta e}{\delta \theta_2} \end{bmatrix} = \begin{bmatrix} r_0 \times (e_{current} - p_0) \\ r_1 \times (e_{current} - p_1) \\ r_2 \times (e_{current} - p_2) \end{bmatrix}$$

$$e = e_{current} - e_{target}$$

The Jacobian matrix is calculated for the system so that we can calculate the inverse and hence the solution. Once the Jacobian system has been defined, we iteratively solve for the change in angle using an approximate technique, such as, Pseudo-Inverse Transpose.

2.6 Summary

We have introduced the basic inverse kinematic concepts, and had a quick refresher course of the kinematics required to develop the algorithms. Your module work should now be structured in such a way that the inverse kinematic simulation is separate from the renderer update, and so that the implementation which you will develop can easily interface with the rest of your system (i.e., modular component programming).

2.7 Tutorial Exercises

Basic Exercises

1. Complete the 3 degrees of freedom example (ik_3dof.cpp)
2. Complete the 1 degree of freedom example (ik_1dof.cpp)
3. Add the option to only move the arm once a solution has been found, Lerp for smoothness. Careful! make sure you add a timeout/max attempts clause
4. Experiment with algorithmic enhancements to see if you can improve the process

Advanced Exercises

- Implement the FABRIK algorithm, as described in the paper [4] and [5]
- Implement an IK solver using the inverse Jacobian method

Chapter 3

Dynamics

3.1 Time

One of the main difference between working out real physics on paper and simulating an interactive simulation is that the world can change and thus the end state is not known in the simulation. For example: given the simple problem of how far will a projectile fly, or how quickly will a box fall, assuming you have all the required parameters (mass, gravity, distances and dimensions), then you can find the definite solution and be 100% accurate. When simulating a rocket flying, or a box falling in a physics engine, there is nothing stopping us using the same equations, finding out the same answer and then interpolating over time. Alas, it's not that simple, for at least two major reasons:

1. The player can alter the parameters during the flight of the box or rocket; they could put something in the flight path, or shoot the box.
2. If we only calculate the end state, simply interpolating between the start and end state will not show certain details. Items in the real would tend to accelerate and decelerate over time, interpolating would not show this accurately.

Fundamentally to build a simulation of real world physics, we must do exactly that: Build a simulator, not a calculator. However, the environment in which our simulator will run (on a CPU) has many small but significant differences to real world physics. The primary issue we will face is time. Relativity aside, time in the real world is continuous, but the time we have in our simulation happens in intervals.
Using some baseline numbers for an average game engine, we usually have 16 milliseconds to do everything we need to calculate the gamestate at one given point in time. After we have done these calculations, we move onto the next 'frame', and then calculate everything for a world that is now 16 milliseconds further on.
This can be a complex concept to grasp, but a crucial one. Every physics 'Tick' we must calculate where everything in the world must be at a single point in time. What this point in time actually is will depend on a few factors. To make matter more complex, games will never run at a constant frame-rate, so how many time the physics must 'tick' to keep pace with an accurate time-line must be well thought through. This section will cover the engineering challenging of deciding when and how much to "Tick"

3.1.1 Simple FPS-Fixed Timestep

The simplest method for when to run a physics tick would be once, every frame. We keep an accurate measurement on the "real time" (variable t), by using the delta-time passed into us.

```
double t;
void render() {...}

void physics_tick(time_now, delta_time){...}

void update(delta_time) {
  do_game_logic();
  physics_tick(t, delta_time);
  t += delta_time;
}
```

This would be fine for only the most rudimentary of physics applications, one tick per frame is not nearly enough, so we can extend this method by doing multiple ticks per frame.

```
#define ticks_per_frame 10
void update(delta_time) {
  do_game_logic();
  for(i < ticks_per_frame)
    physics_tick(t, delta_time / ticks_per_frame);
    t += (delta_time / ticks_per_frame);
  }
}
```

This would work, but we are at the mercy of the framerate. Imagine an extreme example where the renderer took ten minutes to kick out a frame, with our code above, we would be doing one physics tick per minute. Clearly this would cause issues, and so we must use a method that places a limit on the maximum delta-time we use for physics.

3.1.2 Maximum Timestep

```
#define max_physics_tick (1.0 / 600.0)
double t;
void render() {...}

void physics_tick(time_now,delta_time){...}

void update(delta_time) {
  do_game_logic();
  double remainingTime = delta_time;
  while (remainingTime > 0.0) {
    //never have a dt > max_physics_tick
    double dt = glm::min(remainingTime, max_physics_tick);
    remainingTime -= dt;
    t += dt;
    physics_tick(t, dt);
 }
}
```

Here we define a maximum limit to DT (in this case: 600 fps), other than a physics stall (covered later) this method is much more stable and less prone to strange fluctuations in framerate. Having a DT than can vary *might* cause some issues with a physics simulation, and it *may* also difficult to tune for performance. Generally there will be a sweet spot with the number of Ticks or magnitude of DT; having both vary might make life difficult, an issue the next method solves.

3.1.3 Fixed Timestep, non-fixed tick count

```
#define physics_tick_rate (1.0 / 60.0) / 10
double t;
void render() {...}

void physics_tick(time_now,delta_time){...}

void update(delta_time) {
 do_game_logic();
 static double accumulator += delta_time;
 while (accumulator > physics_tick_rate) {
  accumulator -= physics_tick_rate;
  t += physics_tick_rate;
  physics_tick(t, physics_tick_rate);
 }
}
```

This is a subtle change from the previous Maximum Timestep method, this method uses a constant DT, with an accumulator variable that persists across frames. This means that if there is not enough time to do a complete Tick, the remaining time is added to the next frame and an extra Tick could be carried out then. Essentially this means that a tick only happens once a certain fixed amount of time has passed. This allows us to manage the operating load of the physics engine more than the previous example while still maintaining an accurate time line(T).

3.1.4 Game Loop Independant

The previous three methods are attempts to decouple the physics simulation from the framerate of the running application. It may have occurred to you already that this can be done in a much more substantial way: Mutlithreading, i.e keeping the physics simulation running in the background. This would be more of a concurrency task than physics, so it is not covered in this module.

3.1.5 The Tick Objective

What we are really trying to achieve can be summed up in four objectives

- Tick as frequently as possible
- Tick with the smallest DT possible
- Keep the physics simulation time(T) as close as possible to the real time.
- Synchronise with the with game logic with the most upto-date physics data

Physics Stall An issue that you may have come across while playing games; The physics stall (sometime refers to as a death spiral). This happens when the physics system is overwhelmed and cannot process the amount of required ticks before they are due, and thus the entire simulation falls behind. Usually this will also come with a drop in framerate as the renderer is probably equally as overwhelmed, but in a well programmed game loop, the framerate may be fine but the physics will act in a jerky and unrealistic manor. What is happening is that the physics system just cannot keep up, this could be remedied by dropping the tick rate as a last resort, but usually the solution is to limit the amount of physics objects that can ever exist in a scene. For off-line simulations this is not an issue, the simulation will just run longer, but for real-time this is can be a serious problem.

3.1.6 Tutorial Exercises

Exercises

1. Complete the 3 time-step examples (timing_simple.cpp, unfixed_timestep.cpp, unfixed_timestep.cpp)
2. Experiment with limiting the frame-rate and doWork() delay, how does this effect the physics simulation?

Advanced Exercises

- Implement the physics loop in a separate cpu thread.

3.2 Integration

Now that we some of the more technical framework code covered for instigating a physics Tick, we will now look at what actually happens in the physics world during a tick.
Keep in mind that the tick function takes the current time T, and a delta time DT. What we are asking the physics engine to do is calculate where everything will be at time T+DT.

To do this, all the general rules of Newtonian and others are applied to the current state of the world. Unfortunatly, the fact that we are running ticks at discreet points in time can lead to inaccurate results when run over many Ticks. To combat this, different types of *Integration* methods are used.

This is the same theory as integration in the field of calculus, as we are attempting to solve ordinary differential equations in its most basic forms. We are going to hyper-warp over the mathematical proofs that define most of this and stick to two simple concepts: The concept of Order (First order, Second Order...), and the concept of Error.

Order This is the same term (somewhat interchangeable with the term "degree") you have come across when differentiating or integrating equations. First order deals with all variables as they are *"now"*, it does not care for trends, only what the answer to an equation is at one point in time. Second order will deal with the rate of change of one or more variables, this means that more information is required/sampled to get to the final answer. Following this, Third Order will take into account the rate of change of the rate of change, and so on...

Error As we are calculating/integrating at discreet points in time, we will introduce errors. If our timestep was 0 (i.e perfect integration) this would not happen, but this is an impossible task. To put it simply: between our ticks, something may happen, like a quick change in acceleration, or an additional force. We will not see this change until our next tick, and therefore our calculations will be slightly incorrect. A smaller DT will help, but this is a mathematical problem that we just cannot escape. Higher order integration methods can deal with these changes better but never completely 100% accurately.

Impulses and Constraints Building a physics simulation that mimics the real world is our goal, but we also need some extra features; like the ability to apply forces, adjust velocity, teleport items, adjust mass, and add constraints(hinges, axis's, ropes...). These types of things may not play well with the integrator. We will cover this in detail later, but keep in mind the fact that we will need to alter the properties of the items being simulated, and the integrator will have to be able to deal with this.

Let's now look at the most simplest form of Integration to begin.

3.2.1 The Euler method

```
void Tick(const double t, const double dt) {
    for (each object in scene)
    {
        vec3 acc = calculateAcceleration(object);
        object.velocity += acc * dt;
        object.position += object.velocity * dt;
    }
}
```

This looks straightforward and it seems to be just a direct implementations of the laws of motion. This is a First-order Integration method, as it works out the acceleration, velocity and position *right now*. So what is wrong with this method?

- If the acceleration is a function of time or position, and therefore changes over time, we will have missed important data between ticks

In a simple example with only constant forces, I.e., with Gravity ($\approx 10m/s^2$ downwards), the Euler method may be fine. But in a situation with complex forces, things can get inaccurate quickly.

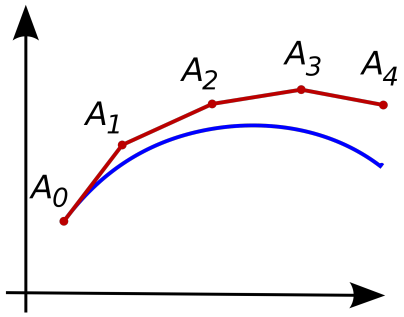

Figure 3.1: **Euler Error Example 1**

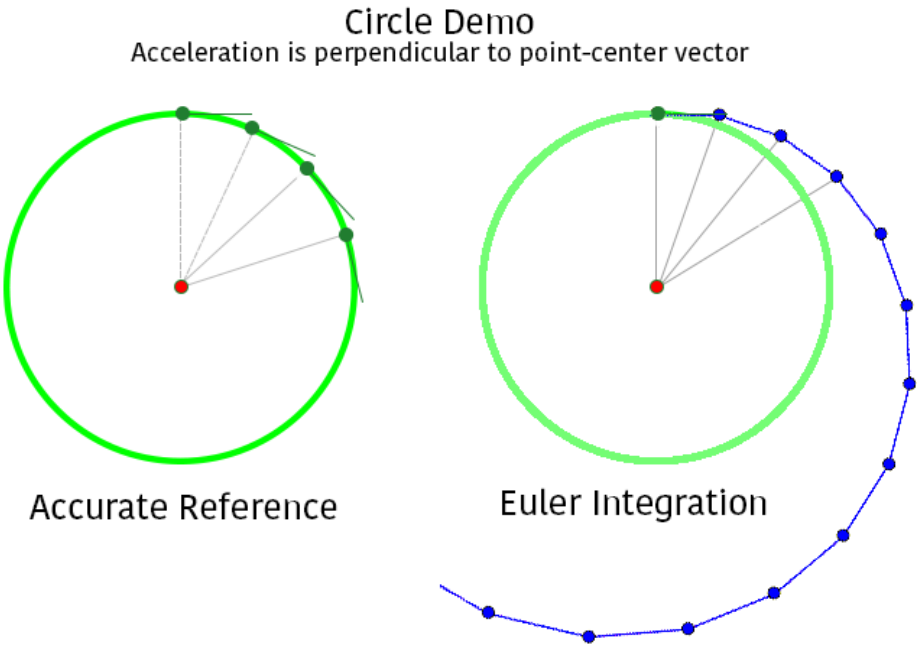

Figure 3.2: **Euler Error Example 2** A smaller timestep would reduce this error, but the effect would be the same, slowly more energy is added to the system

3.2.2 The Verlet method

Verlet is an integration method that was created for use in the Atomic and Molecular Dynamics fields. It is based on a a set of mathematical foundations, principally Taylor Expansion. Fortunately for us, the resulting algorithm is only a short step away for Euler, the short explanation is that we no longer store the velocity of objects, instead we keep track of the previous and current position, and use this to calculate the velocity each Tick.

```
1 void Tick(const double t, const double dt) {
2   for (each object in scene)
3   {
4     vec3 acc = calculateAcceleration(object);
5     vec3 velocity = object.position - object.position_prev;
6     object.position_prev =  object.position;
7     object.position += velocity + acc * dt * dt;
8    }
9 }
```

This small change actually makes this a a second order function, which is much better at conserving internal Energy. Verlet can make solving constraints trivially, as we now do not have to modify velocities, only positions. The downsides it that it requires a significantly smaller timestep to settle on solutions.

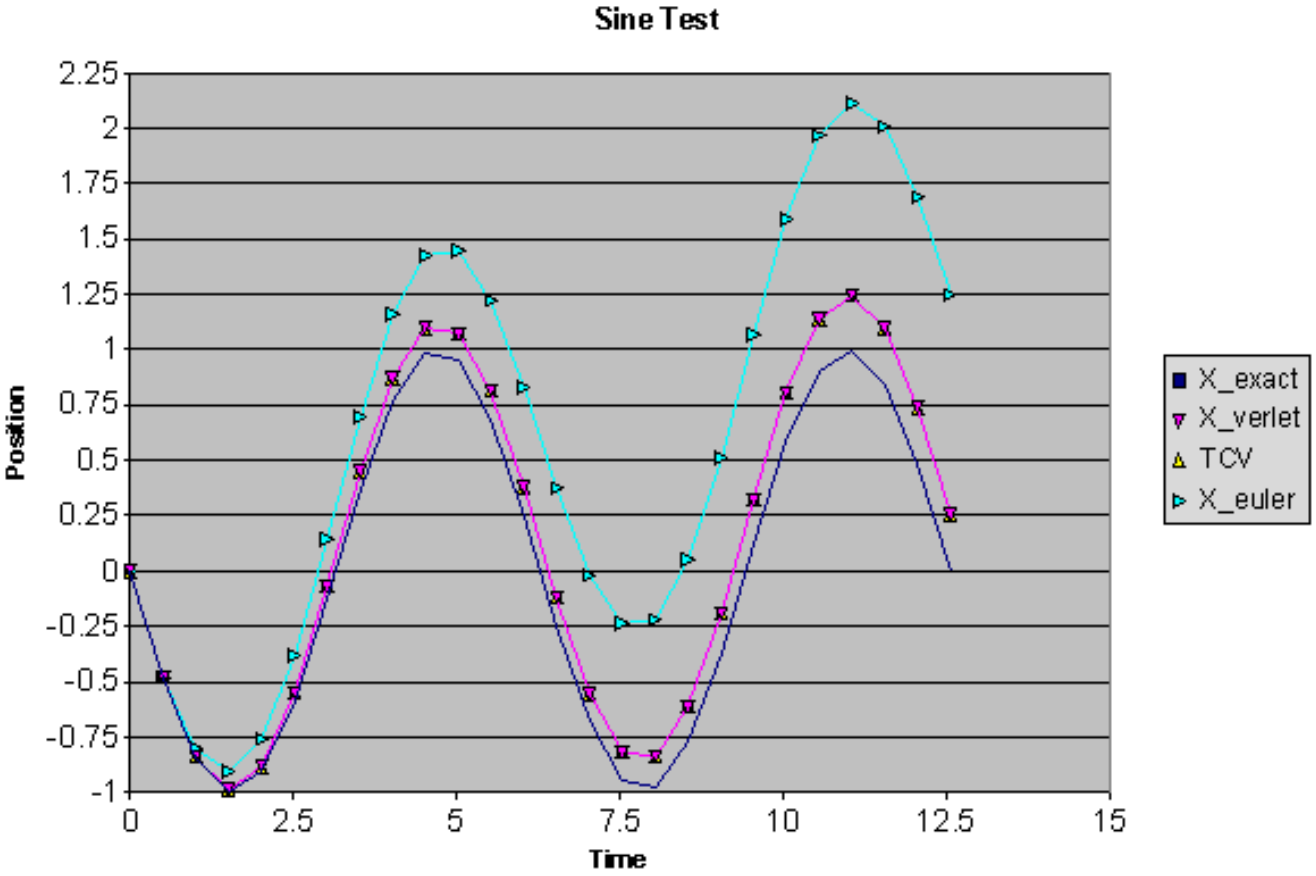

Figure 3.3: **Verlet vs Euler** [1]

3.2.3 The Runge-Kutta (RK4) method

Technically all the integrations mentioned prior to this are versions of Runge–Kutta methods, however the RK4 integrator is usually what is being referred to when Runge-Kutta is mentioned. This is a Fourth order technique and can therefore deal with drastic changes in acceleration more accurately. In laymen's terms, RK4 calculates 4 positions(P) and Velocity(V) spread equally across the timespan of DT. It then sums the results together with a higher weight applied to the middle values than the edges.

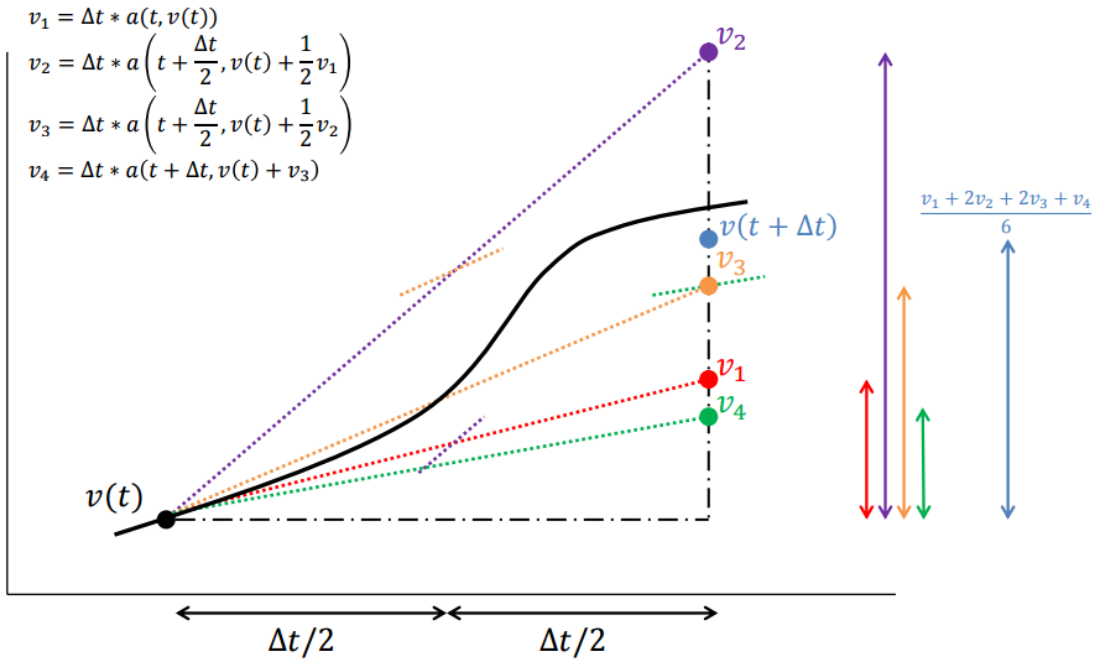

Figure 3.4: **RK4 steps compared**[2] - Black is the ground truth, blue is the RK4 result

```
p1 = position;
v1 = velocity;
a1 = acceleration( p1, v1);
//Calculate P and V, at half dt
p2 = p1 + v1 * dt / 2;
v2 = v1 + a1 * dt / 2;
//Calculate P and V, at half dt, with new V and A
a2 = acceleration( p2, v2);
p3 = p1 + v2 * dt / 2;
v3 = v1 + a2 * dt / 2;
//Calculate P and V, at full dt, with new V and A
a3 = acceleration( p3, v3);
p4 = p1 + v3 * dt;
v4 = v1 + a3 * dt;
a4 = acceleration( p4, v4);
//now we have 4 counts for P and V accross Dt, now "merge" down to one
position += (v1 + 2 * v2 + 2 * v3 + v4) * dt / 6;
velocity += (a1 + 2 * a2 + 2 * a3 + a4) * dt / 6;
```

Listing 3.1: RK4 Pseudocode

3.2.4 Tutorial Exercises

Exercises

1. Complete the 3 integration examples (Euler.cpp, Verlet.cpp, RK4.cpp)
2. Add additional acceleration functions in addition to Gravity
3. Implement a method for measuring the accuracy of each integration method.

Advanced Exercises

- Run all the methods simultaneously (acting on different objects) for visual comparison.

Chapter 4

Particles

4.1 Building up our physics engine

In the previous chapters we have been using standalone "minimal working example" code to demonstrate individual functions. As we move into more advanced techniques, we will need a software harness that is more robust that we can built upon. To this end, you will find a very simple software skeleton of a games engine in the accompanying code. This is a game loop like we have seen before, but with the physics abstracted into it's own corner of the project. Everything is tied together using the Entity Component Model(ECM), which is the de-facto software pattern for games programming.

As we add functionality, we will do this in the form of components. Typical physics components could be something like:

- **Collider** - Notifies when colliding with other colliders
- **RigidBody** - Responds to collision in realistic manor based on shape and mass of object
- **Particle** - May respond to collision, bounces around, probably won't rotate.
- **Forcer** - Adds a Force to everything it touches (Think Wind, Water Current, Air cannon)
- **Kinematic** - Will move the Entity in some way (Think Muscle, Motor, Thruster, Jet Engine), probably communicates with all the constraints on a Ent.
- **Constraint** - Constrains an object based on a set of rules (This could be it's own physics object in it's own right)
- **IK** - Will move in some way to reach target, probably communicates with other IK Ents.

These are just examples common in a fully complete physics engine, some will be combined with others, it really depends on the preferences of the engine designer. For now we will be keeping it simple and creating a basic particle simulation with minimal collision, but we will be coming back to this as our engine get's more featured.

4.1.1 Tutorial Exercises

Exercises

1. Familiarise yourself with the Code, there is nothing to add or fix, but be sure you understand the function of all of it before continuing.

4.2 Creating a simple Particle Simulation

For the purpose of this section we are defining a particle as: "An object with Mass and Size, that can move in all directions but cannot rotate." Usually in games, a particle is something like smoke and sparks; small things that look realistic but can get a way with not being physically accurate. For our simulation we will continue to use ball meshes.

4.2.1 Tutorial Exercises

Exercises

1. Create a simple Ballistic Experiment. Fire a particle in the air, which only subjected to the earth gravitational force (parabolic trajectory).
2. Add aerodynamic friction, either static or dynamic (wind).
3. Have the particle hit a surface (simple collision detection), and react in some way (coefficient of restitution)

4.3 Constraints

Exercises

1. Create a spring constraint between two particles (you can make one unmovable).
2. Test out the spring using different integrators, which is best? how could you test?

Advanced Exercises

1. Create a 2D network of springs and particles, ie. Cloth
2. Create a 3D network of springs and particles, ie. Jelly

Chapter 5

Collision Detection

Chapter 6

Rigid Body

Chapter 7

Appendix

7.1 Bibliography

Bibliography

[1] J. Dummer, "Verlet explained." http://lonesock.net/article/verlet.html. (Accessed on 09/19/2016).

[2] D. N. Pronost, "Game physics." https://liris.cnrs.fr/~npronost/UUCourses/GamePhysics/lectures/lecture%205%20Numerical%20Integration.pdf. (Accessed on 09/16/2016).

[3] What-When-How.com, "Kinematics (advanced methods in computer graphics) part 4," 09 2016.

[4] A. Aristidou and J. Lasenby, "FABRIK: A fast, iterative solver for the inverse kinematics problem," *Graphical Models*, vol. 73, no. 5, pp. 243–260, 2011.

[5] A. Aristidou and J. Lasenby, *Inverse kinematics: a review of existing techniques and introduction of a new fast iterative solver*. University of Cambridge, Department of Engineering, 2009.

7.2 GNU Free Documentation License

GNU Free Documentation License

Version 1.3, 3 November 2008

<http://fsf.org/>

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "**Document**", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "**you**". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "**Modified Version**" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "**Secondary Section**" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "**Invariant Sections**" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "**Cover Texts**" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "**Transparent**" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "**Opaque**".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "**Title Page**" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "**publisher**" means any person or entity that distributes copies of the Document to the public.

A section "**Entitled XYZ**" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "**Acknowledgements**", "**Dedications**", "**Endorsements**", or "**History**".) To "**Preserve the Title**" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange

for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See `http://www.gnu.org/copyleft/`.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright © YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with ... Texts." line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.