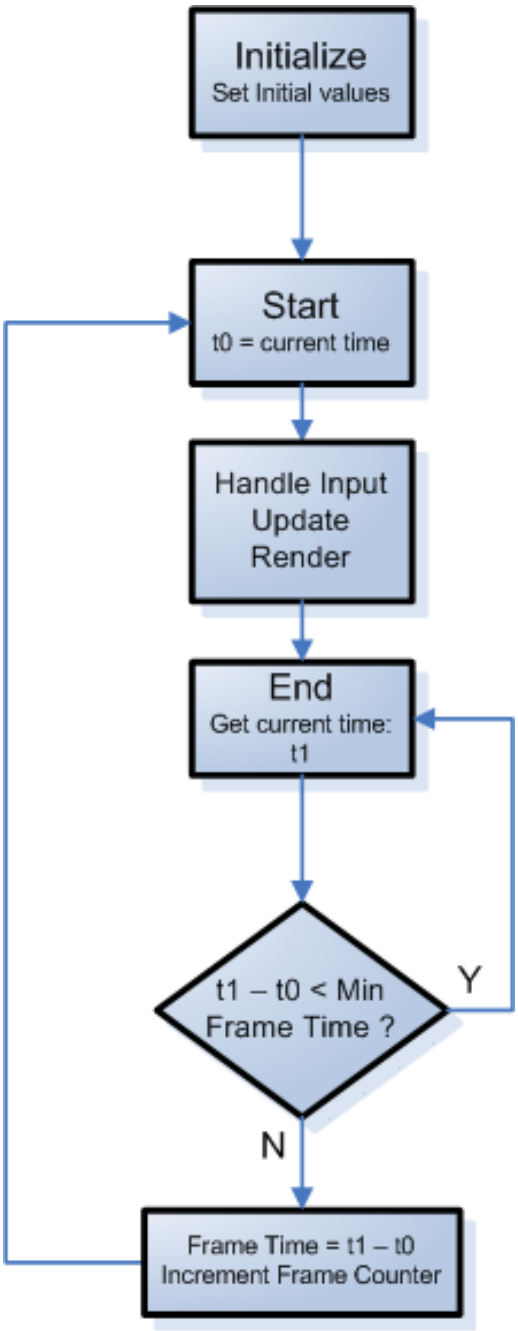
FPS and Time Steps

Previous Method – CSD1130:



Pseudo-Code:

```
GetCurrentTime(startTime);

//GAME LOOP UPDATE

do
{
      GetCurrentTime(newTime);
      deltaTime = (newTime - startTime);

} while(deltaTime < targetFrameTime);
```

What went wrong?

Only when (***deltaTime >= targetFrameTime***), we proceed to the next loop, causing the system to waste time and hang until fulfilling the condition.

However, there is a tiny additional amount of time, ***mt***, wasted between frames, where (***mt = deltaTime – targetFrameTime***).

New Method – CSD2400:

What is a better solution?

A better solution is not to waste time at all and use all the possible time fractions in our simulations.

Answer: Release the FPS!

Pseudo-Code:

```
GetCurrentTime(startTime);

//GAME LOOP UPDATE
GetCurrentTime(newTime);

deltaTime = (newTime - startTime);//computes the actual delta time – no wasting time
```

Advantages:

All systems, based on time, will do their computations correctly, with no time wasted. i.e. the physics system or the animation system…

Possible problem:

Updating the time-based systems will use a non-fixed ***deltaTime*** value from one frame to another. Since every loop might spend different amount of time to finish updating!

Mitigation and solution:

Have a desired fixed target simulated time.

i.e. if you want to simulate your frame time as 60 fps you need to define a targeted fixed delta time as follow:

```
const double fixedDeltaTime = 1.0 / 60.0;//user defined
```

Now all the time-based systems will use this fixed delta time as their input to update their formulas.

i.e. The physics system might need ***fixedDeltaTime*** to update Euler's equations, as follow:

```
//determine the acceleration
acceleration = summedForces * invMass + gravity;
//integrate the velocity
velocity = velocity + acceleration * fixedDeltaTime
//update the position
position = position + velocity * fixedDeltaTime
```

As you can see the physics formula, in the previous example, uses a fixed time step, so that the movement simulation of the different objects looks consistent. Also, this will help when debugging or reproducing an error, because the time steps are fixed and equal, so replaying the simulation must give the same results.

One more essential thing is still missing:

How to synchronize our fixed time step ***fixedDeltaTime*** to the actual time ***deltaTime*** spent by the simulation game loop?

Here's the trick:

You need an additional time variable to compute the accumulated time from one game loop to another. The accumulated time helps in computing the difference between the actual game loop time, and the fixed delta time.

```
double accumulatedTime = 0.0;//one time definition

int currentNumberOfSteps = 0;

void CoreEngine::Update()
{

    currentNumberOfSteps = 0;//reset

    deltaTime = timeEnd - timeStart;//compute actual game loop time

    accumulatedTime += deltaTime;//adding actual game loop time

    while (accumulatedTime >= fixedDeltaTime)

    {

          accumulatedTime -= fixedDeltaTime;//this will store the
          exact accumulated time differences, among all game loops

          currentNumberOfSteps++;

    }


    //Update all the systems here
}
```

Example of its usage in a physics system "***Update"*** function:

```
void PhysicsSystem::Update()
{

    //Loop used in systems that have time-based formula

    for (int step = 0; step < currentNumberOfSteps; ++step)
    {

        //Update Euler's equations and do additional physics
        stuff using fixedDeltaTime instead of deltaTime!

    }
}
```

What does the previous code mean?

Only if ***accumulatedTime*** is greater or equal to ***fixedDeltaTime***, the physics will update.

If the system hangs for some reason, in a way that ***deltaTime*** gets greater from ***fixedDeltaTime*** by N times, then the physics will recover, by updating its formulas N times in one game loop.

If ***accumulatedTime*** is still less than ***fixedDeltaTime***, the physics won't update.

This will keep the physics simulation accurate and similar from one game instance to another and following the fixed delta time simulation specified by the user. The user has control over the speed of the physics simulation by setting a customized **`fixedDeltaTime`** value!

Additional Note:

In the graphics system, turning on VSync, will synchronize the swap buffer rendering with your Monitor's refresh rate.