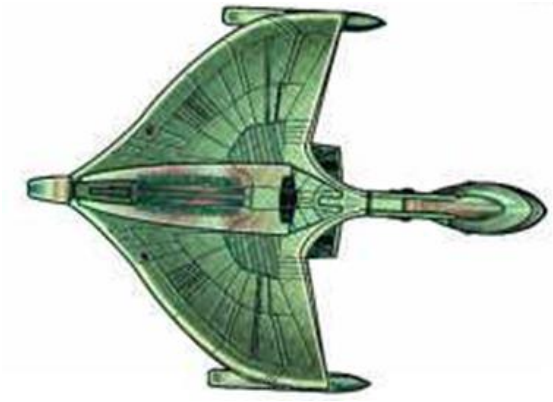
Your Ruber "solar system" and WarBird have been created. The WarBird's camera, movement, and warp capabilities, gravity, missile sites, and intelligens-semita missiles are added to the simulation in this phase. Now it is time for the cadet to move the WarBird, shoot, evade smart missiles, and navigate with gravity. Can the cadet pass flight training, or must they resign? The WarBirdSimulation.xls spreadsheet posted with phase 1 has been updated to describe the parameters of this assignment. Ship and missile speeds are vector forces to travel along the object's "looking at" direction (missile's reference –Z axis, or "in"). The "/sec" values show how far the ship or missile would travel in a second. "Missile updates" is the number of updates the missile exists, its maximum number of update frames, and its maximum distance (w/o gravity). The "distance" values describe the total distance traveled and the time in seconds to travel that distance. The "orbit" columns describe the angle of rotation for each target, and the time to make a complete orbit. The "cadet | TQ" columns show the time quantum values for simulation updates for each cadet type. As the simulation's UPS increases, the play-difficulty of the simulation increases.



Please bring any errors in the simulations parameters to my attention! For any difference between this specification and the posted simulation spreadsheet, the spreadsheet "wins".

**Missile Sites.** A Model for missile sites should be created. They should be easily distinguishable (say a box with different color than the planetoid). The missile sites should be located at the top (highest Y value) of the planetoid. Missile sites should have a bounding radius of about 30.

**Cameras.** The WarBird's camera from the first phase should now move with the ship. When the current view is the WarBird camera, it shows the ship from behind looking over the ship.

**Ship Movement.** The ship's model is now part of the scene. The ship has thrusters that are on or off. We have no-inertial technology – WarBirds can suddenly halt if no thruster is on. (Live with it, it makes an easier assignment to debug and grade.) The ship has positive and negative translation thrusters for its "looking at", "in", or "-Z" reference vector. The ship moves the current ship speed pixels **on** each update when a translation thruster is on. There are three ship speeds: 10 (initial value), 50, 200. Ship speed is selected with the 's' keypress. The ship has rotation thrusters for it's "at", "up", and "right" reference vectors. The ship rotates ± 0.02 radians each update for a rotation thruster. Ship direction or rotation thrusters are set when an arrow key (or key combination) is pressed. Releasing a thruster key (key combination) turns it off. Holding the key down should "fire" many events. The "right" vector is oriented along positive X. The "up" vector is oriented along positive Y. The "looking at" reference vector is oriented along the negative Z. The ship can also "warp" to one of the two planetoid cameras (position and orientation) as a result of the user pressing 'w'. After the ship has warped it can move freely. After warping the ship should be looking at the target, not moving, and see the target move along its orbit somewhat towards the ship. Pressing the 'w' key acts like the 'v' key – the ship warps to the next warp position mod warp positions. The gravity constant only affects the ship's movement (again, live with it). Gravity should initially be set off when the simulation starts.

**Missile Movement.** The intelligens-semita missile model should have a radius of 25. The ship has 9 missiles and each missile site has 5 missiles. A ship missile is fired by pressing the 'f' key. All missiles are smart. A ship missile's initial position is the ship's position and its "looking at" vector is the same as the ship's "looking at". For a Missile Site's missile its position is the Missile Site's position and its "looking at" vector is the Missile Site's "forward", or "looking at" vector (the planetoid's forward direction of rotation). Missile's intelligence tracking is activated after 200 updates. The missile does not detect for

200 updates (not smart, not “live”). All missiles that do not hit their target exist for 2000 update frames. On each update the missile is "live" it attempts to rotate an “orient-to” radians to face its target and makes a move forward (in some frames the missile might not rotate, but should still move forward). You can see when the missile becomes live, because it changes direction. A ship missile’s target is the nearest missile site within detection range. A missile site’s target is the ship if it is within detection range. Why is it a dumb move to fire a ship’s missile when its target is behind the ship? In the ship’s camera view you can watch the missile’s life by turning the ship towards the missile as it moves. Technology advances in the Empire, however the latest version of the intelligens-semita missile often “jumps around” a little but should eventually move towards the target (a nervous, but smart missile). This is acceptable missile behavior for your assignment. However, the empire is always looking for advances in intelligens-semita missile design. Can you advance the Empire’s technology? Some spies have reported that the Federation is investigating quaternions, or, double rotations as a solution to missile "nervousness". The missile site’s missiles are automatically fired when the ship is within a detection distance from the missile. Your instructor is skeptical, but more credit is given to solutions that have smooth (non-nervous) missile tracking. A missile site and the ship can only have one active missile (currentMissile) at a time. When the missile is not targeting (not “live”) it does not test for collisions. Firing a ship’s missile should not cause a collision with the ship on the first update! All missiles disappear (“detonate”) . It’s their existential purpose! They detonate on collision, or when they have travelled all their updates. You do not have to simulate/visualize the explosion. If the WarBird has fired a missile, its missile display count is decremented. The cadet can’t fire (‘f’) another missile until the current missile has detonated.

**TimeQuantum.** Your scene should have a "stepped" update. The update rate is based on a time quantum (TQ) pulse that is used to update object positions. The TQ should be set by user input and have at least 4 values: TQ very large (debug), TQ large (trainee), TQ average (pilot), TQ fast (ace). The game should start in ace (TQ fast). The TQ value (cadet type) can be selected by pressing the ’t’ key. Pressing the 't' key will sequence TQ selections from "ace" to "debug" and then back to "ace". For example, in initial speed (10) with a "pilot" TQ the ship would move forward 25 (1,000/40 updates/sec * 10 / update) units a second (1000 msec.) with no gravity. As in phase 1, the scene is drawn (displayed) as the glutIdleFunc().

**Shaders.** Use the simpleVertex.glsl and simpleFragment.glsl shaders from phase 1. Trust me, phase 3 will make the models look much better!! Of course, you will write the shaders to do that...

**Cadet's actions.**

| Key | Action | Description |
|---|---|---|
| w | warp ship | select next planet warp % nPlanets |
| f | fire | launches available ship missile |
| g | gravity | toggle on / off |
| v | next camera view | select next camera % nCameras |
| x | previous camera view | select last camera used % nCameras |
| t | next TQ value | select next TQ % nTQs |
| s | next ship speed | select next speed % nSpeeds |
| ↑ | ship forward | positive step on "at" vector |
| ↓ | ship backward | negative step on "at" vector |
| → | ship yaws "left" | rotate - 0.02 radians on "up" vector |
| ← | ship yaws "right" | rotate 0.02 radians on "up" vector |
| control ↑ | ship pitches "down" | rotate 0.02 radians on "right" vector |
| control ↓ | ship pitches "up" | rotate - 0.02 radians on "right" vector |
| control → | ship rolls "left" | rotate 0.02 radians on "at" vector |
| control ← | ship rolls "right" | rotate - 0.02 radians on "at" vector |

**Pass or resign.** A cadet passes flight training by destroying both missile sites. Each ship, target, and missile has a bounding sphere for collision testing. If the ship collides with anything the cadet loses the game. If the cadet runs out of missiles without winning, they must resign from the war college. For the ship and missiles, the radius of the bounding sphere is 10 pixels greater than the radius of the object it bounds. All other objects use their radius as a bounding sphere. If the cadet destroys both missile sites, the window title becomes “Cadet passes flight training”. If the cadet resigns the window title becomes “Cadet resigns from War College”.

**Changes to specifications.** You can make "compliant changes" to these specifications. A compliant change is one that corrects any error in the specifications, or makes the program more playable. A "non-compliant" change deviates from the intent of the assignment specifications. We can talk about changes you wish to make in lab before or during the review. You should document all changes in your "readme" report included with your final submission.