1. The environment/world would need to be made more static to reduce the network traffic.  
   The clients would need to communicate changes directly with the network controller and the network controller would then send updates to each client (if that’s not what the game does already).  
   The game would need to make use of hierarchical messaging to give priority to more important messages. The network can make use of spatial subdivision so that clients are only updated about nearby changes. The game could make use of server clusters, where different servers manage different areas of the world, because the game itself consists of many different areas/planets. It would then also need to make use of hot swapping where spare servers are kept on hand to take on excess load when certain areas become overpopulated (too much traffic for just one server).
2. The game could make use of data extrapolation where the difference between the current (lagged) state and the future (true) state is calculated (extrapolated) and slowly corrected over several frames. This avoids the jerky movement as the changes are corrected gradually.
3. The game could (and most probably does) make use of LOD (level of detail) to render images at different levels of detail depending on their distance from the player.  
   Ie. trees which are further away are rendered with less detail and trees which are closer are rendered with more detail.  
   This reduces the processing power required without affecting the players experience because the player likely wouldn’t see the far way details anyway.

The game could make use of several techniques to render the lower LOD images.

The trees that are far away are most likely rendered using simple billboards. These are essentially flat images which are aligned to face the player at all times. Billboards are certainly not convincing close up but can be virtually indistinguishable from a distance and therefore work well for low LOD rendering.  
The medium distance trees or medium LOD images are likely rendered using Image Based Rendering (IMR) or Parallel IBR. Judging from the video I would guess parallel IBR was used. This makes use of several ‘slices’ of images to represent the object (essentially several billboards each representing different sections of the object). This technique requires slightly more processing than a single billboard but can produce much more believable representations from closer distances. However, it is still not entirely believable from up close and is therefore best suited to medium LOD.  
The up close trees (high LOD) most likely make use of speed trees (orthogonal IMR). This method is similar to parallel IBR in that it uses ‘slices’, however it includes slices in at least two directions (this game probably uses three directions to be viewed from above) and makes use of a ‘scan’ of the object as well. This technique can produce very convincing renders even close up. It can also produce multiple unique trees considerably quickly and is thus mostly what they would use for a procedurally generated world.

1. I believe the clouds were also implemented using LOD techniques.  
   It seems to me as though the clouds are rendered using billboard techniques while the player is grounded/far away because I can’t see any movement or dynamic behaviour in the clouds during that time. It also makes sense as this would greatly reduce the processing power required.  
   However, as stated in the question, the clouds clearly appear to have ‘volume’ when the player flies through them. I believe this was achieved using volumetric clouds where the shape/area/geometry of the clouds can be defined and then the ‘cloudiness’ or vapour density can be defined as well. This technique produces very lifelike clouds which can vary in shape and colour to produce different effects. They can also be made interactive so that, for example, the clouds separate when a plane flies through them.
2. A particle system generally consists of a list of particles (each with a predefined parameters defining its appearance and behaviour over time, ie. lifecycle) and a spawn point where the particles are generated from.  
   The spawn point can take different forms/shapes depending on what effect you want.  
   The particles generally have a lifetime from when they are created to when they ‘die’. When a particle ‘dies’ it is not actually destroyed but is rather added to a recycle pool to be reused later. This reduces processing power and makes the particle systems more efficient.

The explosion is most likely created using a combination of parallel particle systems (for the smoke, flames, burning debris, etc.). If the players lasers/bullets are also a particle system (which is possible) then the explosions could be chained particle systems as well, meaning that the laser particles act as immiters or spawn points for the explosion particles if and when required. Alternatively, if the laseres aren’t particles, then the game most likely makes use of raycasting to determine where the player is aiming and where the explosion particle systems should be located.

1. The first issue would be the sheer memory required to keep all of the game components in memory all the time (whether they are being rendered or not).  
   The second issue would be the application of gravity. Naturally you would want each planet to have (or appear to have) it’s own gravitational pull meaning that there would be several gravitational forces in several directions (one for each planet), this is quite difficult to implement.  
   Lastly the basic geometry would be fairly difficult to implement. Ie. from space, the planets appear round, however once in the planet’s atmosphere the planet appears flat. This would also be very difficult to implement.  
     
   My guess is that the game makes use of portals to transition between space and planets. It seems as though each planet has fairly thick clouds at some point in its atmosphere which would be helpful in hiding the transition through the portals.  
   It is likely that the game also makes use of ROAM or Chunked LOD methods to render the terrains, making the game run more smoothly.  
   This portal method would mean that the game doesn’t have to keep all of the game components in memory all the time, it simply has to load the components relevant to the area/planet the player is currently in.  
   It also means that, while in space, the planets can be represented as round but each planet (when passing through the portal) is represented as flat. This would also solve the gravity issue as well as all the planets would simply need a single gravity force pulling downwards (standard gravity).
2. 1. The herd of antelope could make use of a boid algorithm to achieve the flocking behaviour. This algorithm makes use of three primary components (cohesion, separation and alignment) to simulate dynamic flocking behaviour. In this case, the algorithm most likely uses a central controlling object to manage the flock. This allows for flocking combined with fleeing behaviour.  
      When an ‘enemy’ gets too close to the flock (or to a member of the flock) the central controlling object can tell the flock to flee in the opposite direction (while maintaining the flocking behaviour). Certain objects, such as rocks, trees, etc. can also be classified as obstacles (repulsing points) which the flock will avoid so they don’t crash into them.  
      The switching from general flocking to fleeing could be controlled with some sort of state control system but perhaps not. It’s difficult to tell with just the given information.
   2. It seems as though the trees and other obstacles are not avoided by the predator so the AI algorithm would be fairly simple. Most likely, when the predator enters the chasing state (for whatever reason) will locate the flock (or a member of it) and will simply rotate to face that direction. Once facing the flock, the predator will move towards it.  
      Naturally these algorithms will all implement the appropriate animations, speed, particle effects (for the dust) etc. however the basic chase algorithm alone is a simple rotate and move algorithm, likely triggered when the flock is a certain distance away or within a certain region.
   3. The pack/teamwork aspect could perhaps be implemented using a noticeboard technique to coordinate the pack. This way, each member of the pack can send instructions/updates to the rest of the pack by updating the noticeboard. It can then decide what to do next based on the information on the noticeboard.  
        
      Specifically splitting the pack and targeting the remaining prey could make use of influence maps to detect where the two largest sections of the prey are. From there it could trace a path between them to separate them, analys which group is weak and target that group. This process can be repeated until the group of prey is small enough for the pack to attack for the kill.
3. I would choose to make a VR version of the game because the basic premise of the game is other-worldy, meaning that the environment has to appear as though the player is in space or on another planet. The best way to achieve this is with VR where you can completely change the environment that the player sees.  
     
   This would obviously require a VR headset at the very least. I would probably make use of a handheld controller as well. I believe this would allow for more variety of control and actions as these controllers can measure real world movement which provides more input to control the game with. It also allows for more immersion into the game as the controls can vibrate etc. in the players hands, thereby engaging the player’s sense of touch.

Naturally the audio used in the game would also be an important aspect of immersion and making the game more enjoyable.

The user would need different interactions and controls depending on whether the user is flying or grounded, as basic movement would be different. When grounded the user could simply use the controls to move move and could use the headset to actually turn and change directions, whereas when flying, the movement would be limited so that the user can move sideways and has to physically move the controls (in real world) to turn the ship (ie. using the controls like a steering wheel).  
Other controls could remain the same, however, such as aiming and shooting.  
It’s also possible that the user could be required to actually crouch down in order to make the player crouch, and actually has to bend to pick up objects.

1. 1. VR Assignment 1  
      I implemented an outdoor camp sight scene.I made sure to include a lake, mountains/big rocks, hills, trees and grassy areas, to make the scene more interesting. I also decided to set the scene at night simply because I prefer an evening camp environment. The actual camp area had a small hovel, a log as a bench, a fire particle system and some simple camp related items.  
      The interaction included basic movement (walk, run, jump crouch), floating in water, chopping trees into stumps, throwing axes at a target, and adding wood to the fire.  
        
      AI assignment 3  
      I implemented an underwater scene. It included simple terrain with rocks and boulders for obstacles and hiding places. There were different types of plants that would grow, fish which would wander the ocean floor, fish that swarm together and sharks which wander around.  
      All living things have a life cycle (born, grow, die). The fish eat the plants and the sharks eat the swarming fish. The plants reproduce periodically by ‘spreading seeds’. The fish and sharks have a reproduction meter which indicates when they are ready to mate, at which point they find a nearby mate and mate.  
      The floor fish hide from the sharks and the swarming fish flee from the sharks.
   2. I learned that design patterns are very important. My approach to assignment 3 was to implement each behaviour independently (in a separate scene) just to get it working and then to bring it all together. Unfortunately I found that bringing it all together was quite difficult and required a lot of adjustments because I hadn’t implemented them with a single design pattern in mind.
2. The factory pattern is a design pattern used to minimise the processing power required, specifically when creating and destroying objects.  
   It allows for more control (or essentially an override) of the object creation and deletion processes.  
   Generally it makes use of recycle pooling to ‘pool’ objects instead of destroying them and then reuse pooled objects when possible instead of creating new ones. This is because the process of creating and destroying objects is an expensive one.

Essentially the factory pattern allows you to define exactly what happens when an object is created or destroyed and how the program will go about doing it.