Question Final–1: In a basic particle system, what properties does each particle have, and how are these properties updated with each time step (using Euler’s method)?

Question Final–2: What is the advantage of the midpoint method for particle update over Euler’s method? How much better is it?

Question Final–3: How can cloth be simulated using a particle-system approach?

Question Final–4: Modern graphics processors take advantage of two types of parallelism to maximize performance. Describe both of them.

Question Final–5: How is it that modern GPUs, even though they are typically weaker than CPUs, can render graphics more efficiently?

Question Final–6: What is the color quantization problem, and why is it important?

Question Final–7: What is the median cut algorithm for color quantization?

Question Final–8: What is error diffusion as it applies to rendering graphics?

Question Final–9: How is bitBlt’ing helpful for dealing with window scrolling?

Question Final–10: In any graphical system, there must be some technique for the system to notify programs of “events” like mouse clicks, key presses, or window exposure (needing a repaint). Describe the two very different techniques for this mechanism described in class.

Question Final–11: What is novel about the X Windows architecture when compared to other architectures (Microsoft Windows, MacOS 7, etc.)?
2 Solutions

Solution Final–1: (Question, p 1) Each particle has a position and a velocity vector. In Euler’s method, with each time step, the position is translated by some multiple of the velocity, and the current force on the particle is added to the current velocity vector (leading an acceleration to the particle in the next time step).

Solution Final–2: (Question, p 1) The midpoint method approximates the particle’s true trajectory more accurately using the time step. Whereas halving the time step reduces the error of Euler’s method by half only, it reduces the error of the midpoint method to a quarter of the previous error. [Suppose the errors of both are the same ($E$) with a time step of 1, and then we measure the errors with a time step of $1/n$, effectively halving the time step $\log_2 n$ times. We would find that the error of Euler’s method is reduced to $E/2^{\log_2 n} = E/n$, whereas the error of the midpoint method is reduced to $E/4^{\log_2 n} = E/n^2$.]

Solution Final–3: (Question, p 1) We have a two-dimensional network of particles, and a large component of the force on each particle comes from the force computed based on imaginary springs connecting neighboring particles.

Solution Final–4: (Question, p 1) They use “task parallelism” through a pipeline where the rendering task is split into stages of computation, and computations for different locations occur simultaneously in the different stages. This reduces the amount of time any fragment of computational hardware spends idle. They also use “data parallelism” through including multiple pipelines, each handling different data inputs.

Solution Final–5: (Question, p 1) Among the several optimizations used with GPUs: Designers do not waste processor space on circuits that aren’t useful for graphics, most especially cache space; they do not need to worry about making it look like the GPU is processing data serially; and they can take heavy advantage of the very parallel nature of graphics rendering.

Solution Final–6: (Question, p 1) In color quantization, we have an image and some small integer $N$, and we want to approximate it as closely as possible using only $N$ colors. This is mostly useful for reducing the memory requirements for an image, which leads to smaller files and increased transfer time between computers. Historically, it was necessary because older video systems restrict the number of colors displayed on the screen (to reduce the memory costs). Today, it is mostly useful for compressing images on Web screens, most especially for the widely used GIF format, which allows only 256 colors.

Solution Final–7: (Question, p 1) Of the red, green, and blue components we choose the component with the largest range for the input image. Then we determine the median value for that component among the image pixels (where half of the pixels fall below the median, half above), split the pixel set into the two equal portions, and recurse on both halves. We continue the recursion until we have the number of regions allowed.

Solution Final–8: (Question, p 1) Given a grayscale image that we need to render in black-and-white, we go through the pixels in left-to-right, top-down order, and with each pixel if the value for that pixel is more than 128, we color the pixel white (255) and subtract the amount we added to the pixel (255 – level) from the neighboring pixels to be processed in the future. (There are four such neighboring pixels: to the right, down, down and right, and down and left.) If its value is less than 128, we color the pixel black (0) and add the amount we subtracted from the pixel (level) to the future neighboring pixels. Error diffusion generalizes naturally when we can render more than two grayscale levels (but still fewer than the number of levels in the image), and it generalizes to color images (handling each color component as a separate grayscale error diffusion problem).

Solution Final–9: (Question, p 1) If the user scrolls down, then the program can simply tell the hardware to bitBlt the bottom region of the window to the window’s top, and then the program would only need to compute the newly exposed region of the window.
**Solution Final–10:** (Question, p 1) The first, used with Java’s libraries and with GLUT, is the *callback mechanism*. For this, a program tells the graphical system about the location of a routine that should be invoked whenever the event occurs. The graphical system enters this routine at the time it detects the event.

The alternative technique is *event queueing*. the graphical system to provide a call for requesting the next event. Programs written for these systems typically include a “main event loop” that repeatedly calls for the next event and then processes it. This technique is more common than the callback mechanism: It is used in X Windows, Microsoft Windows, MacOS 7, and most others.

[These two techniques correspond to “server-push” and “client-pull” techniques associated with Web pages: With the callback mechanism, the graphical system pushes the event into the program; and with event queueing, the client pulls the event from the graphical system.]

**Solution Final–11:** (Question, p 1) X is not part of the operating system — it runs independently as a process on top of the system. Processes can display graphics by sending messages to the process running X, which then translates this into messages for the graphics hardware.

Another difference is that X Windows has a very minimalistic approach: The tasks it performs are only basic graphics operations (like drawing rectangles and text, or allocating a region of the screen as a window, and decisions concerning GUI design are left to other processes to manage. Design decisions left out of X include such fundamental things as window management, menu display. This means increased flexibility at programs, although it comes at the expense of programs being less standardized.