ESE5320: System-on-a-Chip Architecture

Day 15: October 24, 2022
Development by
Incremental Refinement

Today

- Incremental Refinement
  - Demand
  - Benefits
  - Simplifications (Part 2)
    - Example: render
  - Interfaces (Part 3)
  - Defensive Programming
- Source Code Repositories

Message

- Focus on interfaces early
  - Integrate first
- Start with something simple that works end-to-end and incrementally refine
  - May lack features
  - May perform poorly
  - …but it lets you resolve interfaces early

Project Planning

- What is more likely to happen to the part of a project you leave to the end?
- Why might it seem natural to leave integration of components to the end of a project?
  - After fully developing components

Common Mistake

- Build pieces, then integrate at the end
- Spend most of available time on components
  - Then try to integrate for first time near deadline
  - Not enough time to integrate/debug at end
    - Worst-case don't have a working solution
    - Spend more time fixing than if had identified incompatibilities early

Standard Chip Aphorism

- Almost all ASICs work when first fabricated
  - …until you put them on the board.
    - Then maybe 50%
- [usually say "first spin" – where each "spin" is a separate manufacturing run]
- ASIC: Application Specific Integrated Circuit
  - (custom chip)

Recommended Approach

- Decompose problem
- Focus on how components interact
- Figure out simplified functionality easy to assemble
- Get minimum functionality end-to-end system running early
  - Even if means cut corners, solve simplified piece of problem
- Chart path to refine pieces to goal

Benefits

Early Integration

- What benefits might get from integrating early?

Benefits: Overview

- Agree on interfaces up front
- Supports parallel development, testing, debugging
- Confidence-boosting win of having something that works
- Digest problem -- supports work in small bursts

Interface First

- Agree on interfaces up front
- Each component knows interface
- Can replace each component independently
- Simple baseline provides scaffolding
  - Run end-to-end tests
  - Test components in context

Parallel Development

- With interfaces defined…
- Each component can be (mostly) independently developed and refined
- Simple baseline provides scaffolding
  - Framework to test each component independently as develop and refine
- Particularly important for team
  - …helpful for individual, too
    - Contains what need to think about at a time

Confidence Boost

- Get to see it working
- Know you have something
  - Just a question of how sophisticated can you make it?

13

Digested Problem

- Easier to concentrate on what need to do for this piece
- Can make tangible process in short bursts
  - …time can find between lectures…

14

Continuous Integration

- Pieces always fit into interface scaffold
- Add pieces, functionality as available
- See improvement
- Identify interface problems early
  - …and refine them

15

Part 2: Example

Rendering

16

Rendering Example

- Create a 2D (video) image of a 3D object (set of objects)
- For: computer-generated graphics
  - Movies
  - Video games

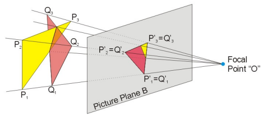
17

Rendering

- Input:
  - collection of triangles (with color)
    - Each 3 (x,y,z) positions
  - Viewpoint
    - Another (x,y,z) point
- Output
  - 2D raster image (what you see on screen)
    - Showings what's visible
      - Some things will be hidden behind others

18

Rendering Decomposed



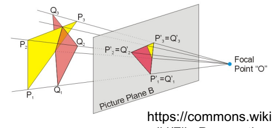
- Pipeline of
  - Projection
    - Where do the points of this triangle end up in the viewed image?
    - Matrix-multiplication to translate points

https://commons.wikimedia.org/wiki/File:Perspective_Projection_Principle.jpg

Penn ESE5320 Fall 2022 -- DeHon 19

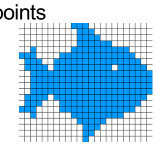
19

Rendering Decomposed



https://commons.wikimedia.org/wiki/File:Perspective_Projection_Principle.jpg

- Pipeline of
  - Projection
    - Where do the points of this triangle end up in the viewed image?
    - Matrix-multiplication to translate points
  - Rasterization
    - Turn into pixels
    - Fill pixels for triangle

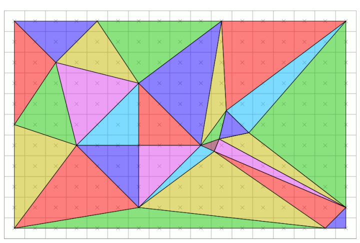


https://commons.wikimedia.org/wiki/File:Raster_graphic_fish_20x23squares_sdtv-example.png de:Benutzer Diskussion:Andreas -horn- Hornig/Bildwerkstatt

Penn ESE5320 Fall 2022 -- DeHon 20

20

Rasterization

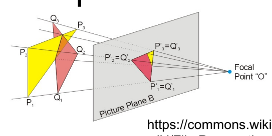


By Drummyfish - Own work, CC0, https://commons.wikimedia.org/w/index.php?curid=80204437

Penn ESE5320 Fall 2022 -- DeHon 21

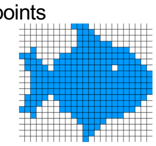
21

Rendering Decomposed



https://commons.wikimedia.org/wiki/File:Perspective_Projection_Principle.jpg

- Pipeline of
  - Projection
    - Where do the points of this triangle end up in the viewed image?
    - Matrix-multiplication to translate points
  - Rasterization
    - Turn into pixels
    - Fill pixels for triangle

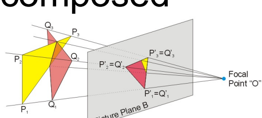


https://commons.wikimedia.org/wiki/File:Raster_graphic_fish_20x23squares_sdtv-example.png de:Benutzer Diskussion:Andreas -horn- Hornig/Bildwerkstatt

Penn ESE5320 Fall 2022 -- DeHon 22

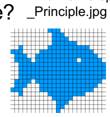
22

Rendering Decomposed



https://commons.wikimedia.org/wiki/File:Perspective_Projection_Principle.jpg

- Pipeline of
  - Projection
    - Where do the points of this triangle end up in the viewed image?
    - Matrix-multiplication to translate points
  - Rasterization
    - Turn into pixels
    - Fill pixels for triangle
  - Z-buffer
    - Keep only the ones on top (not hidden)
      - 2D image + Z-depth – keep smallest



https://commons.wikimedia.org/wiki/File:Raster_graphic_fish_20x23squares_sdtv-example.png de:Benutzer Diskussion:Andreas -horn- Hornig/Bildwerkstatt

Penn ESE5320 Fall 2022 -- DeHon 23

23

What's Hard (Preclass 1)

- What's hard about each part?
  - Projection?
  - Rasterization?
  - Z-Buffering?

Penn ESE5320 Fall 2022 -- DeHon 24

24

Simplifications

Simplification: Overview

- Solve simpler problem
- Handle special subset of cases
  - Avoid hard corner cases
- Don't worry about performance
- Placeholder – stand in for real task
  - Do minimal thing
  - Use existing code

Simple Placeholder

- Identity function work?
  - Pass input to output
- Get form right in simple way?
  - E.g. compression
    - Drop samples/images/pixels to get down?

Simplify (Preclass 3)

- How could we simplify
  - Projection?
  - Rasterization?
  - Z-Buffering?

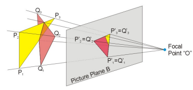
Simplified Projection Example

- Projection as identity function?
  - Will definitely give wrong image
    - Except when viewpoint 0,0,0….
      And all triangles at same depth…
  - But the output of projection is triangles
    - …so has right form for communication

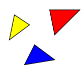
Simplified Rasterization

- Maybe: Just put output pixels for triangle corners?
  - Definitely wrong
  - Has right form

Simplified Z-Buffer Example

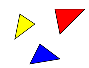


- Intended
  - Z-buffer
    - Keep only the ones on top (not hidden)
      - 2D image + Z-depth – keep smallest
- Simplified
  - Just keep last value given
  - If nothing overlaps → correct
    - test with non-overlapping objects
  - Even if overlap
    - Looks wrong, but data has correct output form



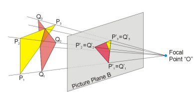
Solve Subset

- Are there cases that are easier and cases that are harder?
  - Can arrange input/tests to only include easier cases first
- Solve the simple cases first
  - E.g. non-overlapping objects in Z-buffer
- Add support for harder cases later



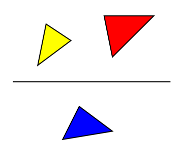
Data Parallel

- How exploit data parallelism in projection?



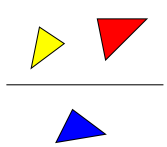
Parallel Rendering Example

- Exploit data parallelism in rasterization
  - Cut image into pieces
    - Simplest: top half, bottom half
  - Separate threads to rasterize each piece



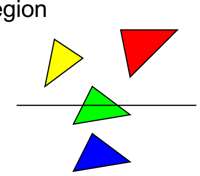
Parallel Rendering

- Maybe ideal: rasterization sends triangle to appropriate rasterization thread
  - If in top half
    - send to top
  - Else
    - Send to bottom
- What could make hard?



Parallel Rasterization

- Simple
  - Triangles exclusively in one region
    - One half
  - Send to appropriate half
- Hard
  - Triangle in both halves
    - Send to all (both)
    - Or compute what goes in each and send triangles to each



Parallel Rasterization Refinement

- Start simple
  - Assume only in one half, and only send there
  - Use test cases split by halves
- Incrementally get more sophisticated
  - Sometimes send to both
- Incrementally more
  - Compute triangles for each region

What makes hard?

- Can avoid that on initial pass?
  - E.g. – avoid computing what part of triangle is in each region

Solve Small Instances?

- If challenge is scale (handling large problems)
  - Solve small problems first
  - E.g. work on 64x64 image
    - If trying to hit real time, easier with small image
    - Small image may fit in BRAM (on-chip memory)
      - Avoid complexities of data movement initally

Non-Optimized Implementation

- Often complexity comes from optimized implementation
  - Start with simplest, non-optimized version as placeholder
  - E.g.
    - Brute force solution instead of clever algorithm
      - Perhaps my most common mistake
    - Large, inefficient data structure
      - Instead of a more complicated, compact one

Window Filter

Day 14

- Compute based on neighbors
- for loop:

```
for (y=0;y<YMAX;y++)
  for (x=0;x<XMAX;x++)
    o[y][x]=F(d[y-1][x-1],d[y-1][x],d[y-1][x+1],
              d[y][x-1],d[y][x],d[y][x+1],
              d[y+1][x-1],d[y+1][x],d[y+1][x+1]);
```


Window Filter

Day 14

- Single read and write from dym, dy
- for loop:

```
for (y=0;y<YMAX;y++)
  for (x=0;x<XMAX;x++) {
   dypxm=dypx; dypx=dnew; dnew=d[y+1][x+1];
   dyxm=dyx; dyx=dyxp; dyxp=dy[x+1];
   dymxm=dymx; dymx=dymxp; dymxp=dym[x+1];
    o[y][x]=F(dymxm,dymx,dymxp,
              dyxm,dyx,dyxp,
              dypxm,dypx,dnew);
    dym[x-1]=dyxm;dy[x-1]=dypxm; }
```

Software First

- Functional placeholder in software first

43

Leverage Existing Solutions

- Run some existing package, library to get the right answer
  - E.g.
    - call MATLAB to solve a matrix
    - Invoke unix sort routine to get sorted data
    - Invoke stand-alone image compressor or renderer

44

What components depend upon?

- Can a component output any data (random data?) and be adequate to exercise components interacts with?
  - E.g. if feed into an integrator/accumulator
- Need to output data of a given size?
- Output need to maintain some property?
  - Sorted?
  - Unique?
- Is it ok if doesn't do its intended job well?
  - E.g. intended to compress…

45

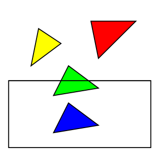
Interfaces

Part 3

46

Division of Task

- Who is expected to do what?
  - E.g.,
    - Which piece discards duplicates?
    - Which piece removes/flags invalid input?
  - E.g. Renderer
    - Does Projection only send in-bound triangles to each region rasterizer?
    - Or does each region rasterizer need to deal with out-of-bounds triangle coordinates?



47

Need to Know

- What information does each component need to know?
- How do we get that information to each component?

48

Rendering Interface (Preclass 4)

- What need to communicate between
  - Projection → Rasterization
  - Rasterization → Z-Buffering

3D Rendering: Need to Know

- Projection
  - How many triangles (int)
  - Triangle points (x,y,z) triples (float)+ color (short)
  - Viewpoint x,y,z (float)
- Rasterization
  - How many triangles (for region)
  - Triangle points (x,y,z) triples + color (short)
  - When done
- Z-buffer
  - (x,y,z,color) points – short
  - How many (when done)?

How Communicate?

- Arrays
- Streams
- Shared memory locations?
- Variable lengths?

3D Rendering

- All naturally streaming
- All potentially variable
  - Number of triangles depend on object complexity and number of objects
  - Projected triangles depend on number in each region
    - Not know in advance
  - Pixels sent depends on size of projected triangles which changes with viewpoint
    - Not know in advance

3D Rendering

- Triangles and pixels unknown up front
- How might we communicate number of triangles/pixels – communicate when done?

3D Rendering

- Triangles and pixels unknown up front
- How communicate?
  - Send a record that means end-of-image?
    - Extra bit?
    - ```
      struct send_triangle {
                      short p1x,p1y,p1z,
                            p2x,p2y,p2z,
                            p3x,p3y,p3z,
                            color;
                    Boolean last; }
      ```
    - 161b
      ```

3D Rendering

- Triangles and pixels unknown up front
- How communicate?
  - Send a record that means end-of-image?
    - Extra bit?
  - Send in blocks with maximum size
    - Accompany each block with a length
    - Length is a separate stream from data
    - For(i=0;i<TRIANGLES;i+=5)
      - block_size.write(5);
      - For(j=0;j<5;j++) triangles.write(t[i+j]);
    - If (i!=TRIANGLES)
      - block_size.write(TRIANGLES-i);
      - for(j=0;j<TRIANGLES-I;j++)

Properties components can assume?

- Sorted?
  - If Z-buffer could assume sorted
    - Just keep first at location (last if decreasing)
- Non–duplicate?
- All in-bound?
- Bound on input size in a block?

Interfaces May Change

- Interface first
  - Means less surprise later
  - Doesn't mean know everything up front
- Experience making simple work … and refining simple
  - Often best way to understand needs of problem
- Refine the interfaces incrementally, too

3D Rendering Start

- Might start
  - Projection = identity (convert short)
  - Rasterization = triangle corners
  - Z-buffer = save last
  - Connect with streams
    - Streams data has one bit for last triangle, pixel
- Can put together quickly

Rendering Start Placeholder

```
for(int i=0;i<TRIANGLES;i++)
     struct triangle2d t2d;
     t2d.p1x=tr[i].p1x;
     t2d.p1y=tr[i].p1y;
     t2d.p1z=tr[i].p1z;
     // same for p2, p3
     t2d.color=tr[i].color;
     t2d.last=(i==TRIANGLES-1);
     rasterize_in.write(t2d);
```


Rendering Start Placeholder

```
While (true)
  rt2d=rasterize_in.read();
  pt.x=rt2d.p1x; pt.y=rt2d.p1y; // and z
  pt.last=false; pt.color=r2d.color;
  zin.write(pt);
  pt.x=rt2d.p2x; pt.y=rt2d.p2y; // z
  pt.last=false; pt.color=r2d.color;
  zin.write(pt);
  pt.x=rt2d.p3x; pt.y=rt2d.p3y; // z
  pt.last=tr2d.last; pt.color=r2d.color;
  zin.write(pt);
  if (tr2d.last) break;
```

Rendering Start Placeholder

```
while (true)
  zpt=zin.read()
  image[zpt.y][zpt.x]=zpt.color;
  if (zpt.last) break;
```


Rendering Start Refine

```
while (true)
  zpt=zin.read()
  if (z[zpt.y][zpt.x]>zpt.z)  {
     image[zpt.y][zpt.x]=zpt.color;
     z[zpt.y][zpt.x]=zpt.z;    }
  if (zpt.last) break;
```


Rendering Start Refine

```
// initialize z[] to MAXVAL
while (true)
  zpt=zin.read()
  if (z[zpt.y][zpt.x]>zpt.z)  {
     image[zpt.y][zpt.x]=zpt.color;
     z[zpt.y][zpt.x]=zpt.z;    }
  if (zpt.last) break;
// large image – may need to split?
//          … move off chip?
//          represent in clever way
```


3D Rendering Independent Refinement

- Projection – actually calculate projected coordinates
- Rasterization – calculate pixels per triangle
  - Test just fine using identity from projection
- Z-buffer – add in Z-ordering
  - Also testable with placeholder results

3D Rendering Refinement

- Put them back together and work with interface defined
- Could decide to change to communicating with blocks
- Could refine for parallel rasterization
  - …and could do that in pieces

Defensive Programming

Validate Assumptions/Requirements

- If require a property on input of a module
  - Good to have (optional) code to test for it
  - [add that code second]
    - Adds code/complexity to check
    - E.g. check actually is in-bounds if should be
  - Condition it in #ifdef so can disable for production, and re-enable for debug
  - Good to catch invalid assumptions early
    - …rather than spend time debugging to discover
    - Setup discussion about interface…which part got it wrong

67

Swap Modules

- Make it easy to swap out implementations
  - Swap between placeholders and refined implementations
  - Swap among implementation versions
  - Good to understand where problems introduced

68

Source Code Repositories

git, svn

69

Repository Message

- When working on a project, especially with other people, want to use a source code repository
- We've encouraged you to use for HWs
- Start one for project group as soon as you create a project team

70

Basic Idea

- Central authoritative home for code
  - Everyone can access
    - Even if someone gets sick, laptop crashes
- Keeps track of all versions
  - As iterate and refine
- Maybe keep track of multiple, in-use versions at once → branches

71

Basic Benefits

- Keep organized
  - Common place for everything
- Keep track of history
  - Can go back to previous versions
    - If screw up; if thought worked before
    - Lowers chance of accidentally deleting
    - …or losing when laptop disk crashes
- Able to work on independently
  - Share/integrate as stable
- Branches
  - Experiment without breaking main version
  - E.g. change an interface

72

Big Ideas:

- Integrate first
  - Focus on interfaces early
- Start simple
  - Something that works end-to-end
- Improve incrementally and iteratively

Admin

- Feedback
- Wednesday: Project out and introduction
- HW7 due ~~Friday~~ Sunday