CS35L Software Construction Laboratory

Lab 5: Sneha Shankar

Week 7; Lecture 2

Basic pthread Functions

There are 5 basic pthread functions:

1. pthread_create: creates a new thread within a process

2. pthread_join: waits for another thread to terminate

3. pthread_equal: compares thread ids to see if they refer to the same thread

4. pthread_self: returns the id of the calling thread

5. pthread_exit: terminates the currently running thread

pthread_create

- **Function:** creates a new thread and makes it executable
- Can be called any number of times from anywhere within code
- Return value:
  - Success: zero
  - Failure: error number

Parameters

```
int pthread_create( pthread_t *tid, const pthread_attr_t *attr,
                    void *(my_function)(void *), void *arg );
```

- **tid**: unique identifier for newly created thread
- **attr**: object that holds thread attributes (priority, stack size, etc.)
  - Pass in NULL for default attributes
- **my_function**: function that thread will execute once it is created
- **arg**: a *single* argument that may be passed to my_function
  - Pass in NULL if no arguments

pthread_create Example

```
#include <pthread.h> …
void *printMsg(void *thread_num) {
		int t_num = (int) thread_num;
		printf(“It’s me, thread %d!\n”, t_num);
Return NULL;
 }

int main() {
		pthread_t tids[3];
		int t;
		for(t = 0; t < 3; t++) {
				int ret = pthread_create(&tids[t], NULL, printMsg, (void *) t);
				if(ret) {
						printf(“Error creating thread. Error code is %d\n”, ret”);
						exit(-1); }
		}
}
```

Possible problem with this code? (Hint: use pthread_join)

If main thread finishes before all threads finish their job -> incorrect results

pthread_join

- **Function:** makes originating thread wait for the completion of all its spawned threads' tasks
- Without join, the originating thread would exit as soon as it completes its job
  - ⇒A spawned thread can get aborted even if it is in the middle of its chore
- Return value:
  - ⇒Success: zero
  - ⇒Failure: error number

Arguments

```
int pthread_join(pthread_t tid, void **status);
```

- **tid**: thread ID of thread to wait on
- **status:** the exit status of the target thread is stored in the location pointed to by *status
  - Pass in NULL if no status is needed

pthread_join Task 1

Write a c program to solve the previous problem we saw in pthread_create example

Task 1 solution

```
  for (t=0; t<3;t++) {
        int ret1 = pthread_join(tids[t], NULL);
        if(ret1) {
        printf("Error joining thread. Error code is %d\n", 
ret1);
        exit(-1);
        }
  }
```

Task 2

Create a C program to increment two variables x and y from 0 to 100 using two different threads. Print the new values once both threads have incremented.

Hint: main = 1 thread

Task 2 solution

```
void *inc(void* x) {
int *x1= (int *)x;
while(++(*x1) < 100);
printf("x is incremented\n");
return NULL;
}

int main() {
int x=0,y=0;
pthread_t t1;
if(pthread_create(&t1, NULL, inc,
&x)) {
fprintf(stderr, "Error creating
thread\n");
return 1;
}
while(++y < 100);
printf("y increment finished\n");
if(pthread_join(t1, NULL)) {
fprintf(stderr, "Error joining
thread\n");
return 2;
}
printf("joined\n");
return 0;
}
```

Lab 6

- Evaluate the performance of multithreaded sort command
- Delete empty line
- Add /usr/local/cs/bin to PATH (export)
- Generate 10M random single precision floating point numbers
  - /dev/urandom pseudo-random number generator
  - od -An -t fF -N size < /dev/urandom
  - Find out about each of these options

Lab 6...

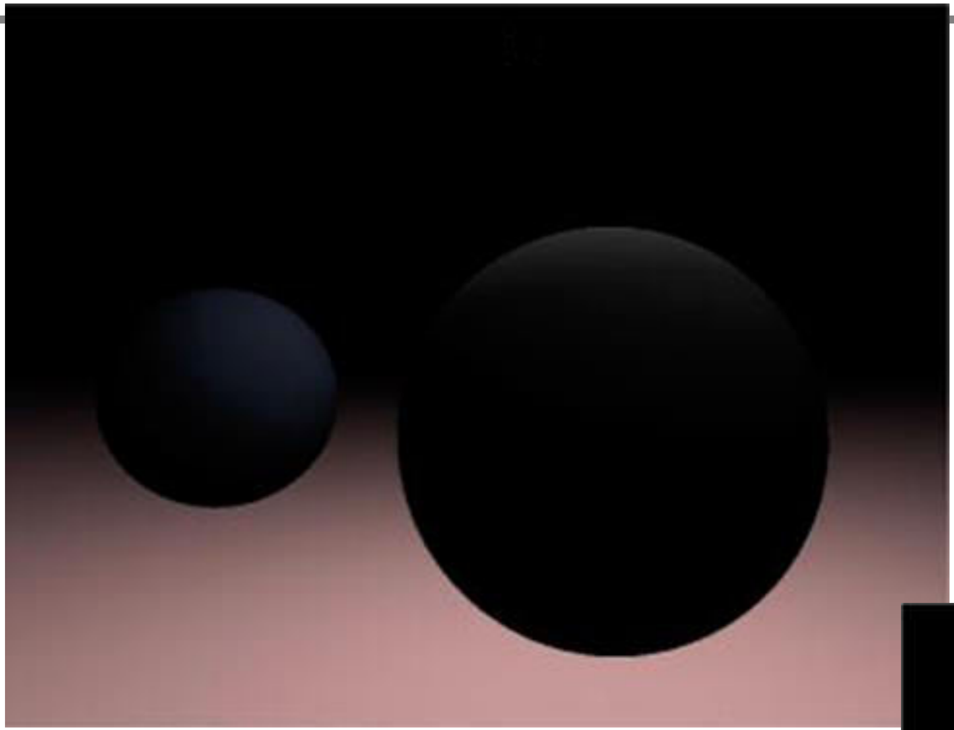
- od: writes contents of its input files to stdout in a user specified format
- Options:
  - -t fF: single precision floating point
  - -N count: Format no more than count bytes of input
- sed, tr: remove address, delete spaces, add newlines between each float instead of ‘ ‘
  - generate random numbers | tr -s ____?____ > txt.file

Lab 6...

- use time -p to time ‘sort’ -g on generated data
- Send output to /dev/null (to dispose unwanted output streams)
- run sort with --parallel to specify thread count and -g option: compare by general numeric value
  - use time to record sort time for 1,2,4,8 threads
  - time - p /usr/local/cs/bin/sort -g --parallel=2 txt.file > /dev/null

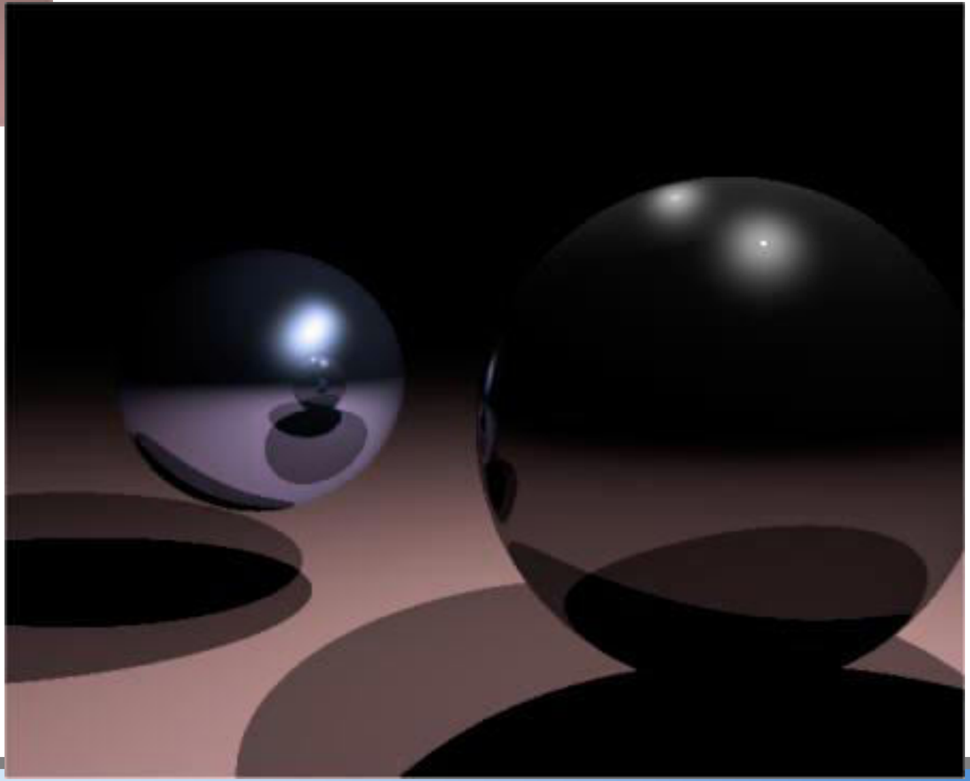
Ray Tracing

- An advanced computer graphics technique for rendering 3D images
- Mimics the propagation of light through objects
- Simulates the effects of a single light ray as it’s reflected or absorbed by objects in the images



Without ray tracing

With ray tracing



Computational Resources

- Ray Tracing produces a very high degree of visual realism at a high cost (yields high quality rendering)
- The algorithm is *computationally intensive*
- Good candidate for multithreading (embarrassingly parallel)
  - Threads need not synchronize with each other, because each thread works on a different pixel

Homework 6

- Download the single-threaded ray tracer implementation
- Run it to get output image
- Multithread ray tracing
  - Modify main.c and Makefile
- Run the multithreaded version and compare resulting image with single-threaded one

Homework 6

- Build a multi-threaded version of Ray tracer
- Modify “main.c” & “Makefile”
  - Include <pthread.h> in “main.c”
  - Use “pthread_create” & “pthread_join” in “main.c”
  - Link with –lpthread flag (LDLIBS target)
- make clean check
  - Outputs “1-test.ppm”
  - Can’t see “1-test.ppm”
    - sudo apt-get install gimp (Ubuntu)
    - X forwarding (lnxsrv)
      - ssh –X username@lnxsrv.seas.ucla.edu
    - gimp 1-test.ppm

Tips

- Ensure no compile error exists!
- Read the source code to understand the task
- Don’t modify other functions in the original code
- Submit a gzipped file .tgz
- Keynote: How to divide the task to run multiple threads
- Difficulty: the 3rd and 4th arguments of pthread_create function
  - Argument 3: a function that divides the input by threads
  - Argument 4: an array to hold data for each thread

1-test.ppm

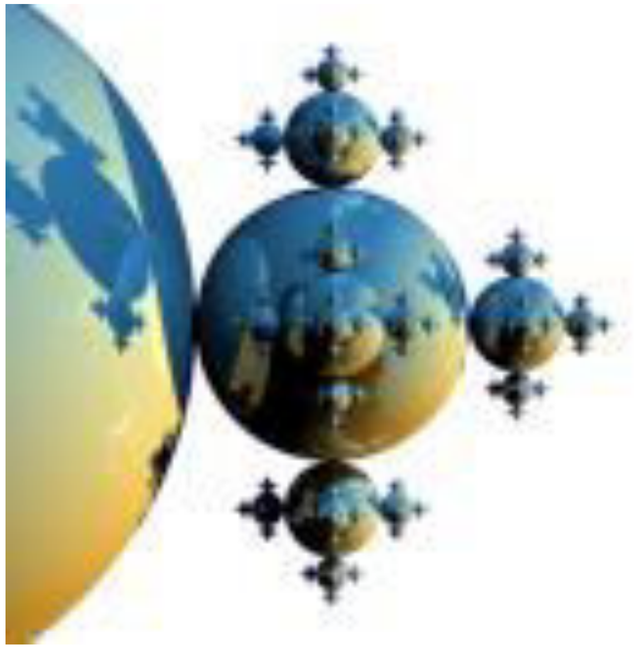


Figure. 1-test.ppm & baseline.ppm