APPENDIX A

COMPUTER SOFTWARE ROUTINES

A.1 Overview

The following programs were written in C and compiled using the Microsoft C/C++ version 7.0 compiler. A routine originally written in FORTRAN by C. Ding to control the movement of the translation stages using PCLAB routines from Data Translation was rewritten in C to allow the stages and the frame grabber to be controlled using the same program. The stages were driven using a DT 2801 D/A card. The routines to operate the frame grabber and frame processor cards (DT 2851 and 2858) were from the DT-IRIS routine library from Data Translation.

Included in this appendix are:

A.2 Image Acquisition Code: APRUN.C

A.3 Data Reduction Code: DETAIL2.C

A.2 Image Acquisition Code: APRUN.C

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <isdefs.c>
#include <time.h>

extern short input_digital_on_trigger( short, unsigned short, unsigned short * );
extern short enable_for_output( short );
extern short output_digital_value( short, unsigned short, unsigned short );

void sm(long int na, int nb, int nc, int ii);

main()
{
  FILE *fp;
  int sync, display, frame, frame2, frcnt, enc, reso, id, jd, dlay;
  int brt, i, j, k, jj, cuts, conary[20], istat, nb, nc, ii;
  int row0, hei, col, wid, lines[512], avary[512];
  unsigned int *regptr, regcon, regmask, regout;
  char desig[6], names[100][13], syscall[35], *nptr, gostrng[5];
  char hdrfile[11];
  float numin, delta, startpos;
  div_t namer;
  time_t timer1, timer2;
  double dtime;
  long int na, isum, minsum;
  short stat, podv;
  unsigned short podv2, podv3, t4, t5;

/*  Initialize variables  */

  minsum = 100000;
  regmask = 32;
  podv = 1;
  podv2 = 0;
  podv3 = 0;
  t4 = 255;
  t5 = 136;
  col = 0;
  hei = 1;
  wid = 512;
  nb = 0;
  numin = 0;
  nc = 0;
  enc = 0;
  reso = 1;
  brt = 0;
  frame = 0;
  frcnt = 1;
```

```
  ii = 2;
  for(i=0;i<100;i++)
    for(j=1;j<13;j++)
      names[i][j]= 0;
  for(jj=0;jj<512;jj++)
    avary[jj] = 0;

/*  Move stage to starting posn  */

  printf("Use which stage? (0, 1, or 2): ");
  scanf("%i",&nc);
  do
  {
    printf("Move stage up(0) or down(1)? ");
    scanf("%i", &nb);
    printf("Move how many millimeters? ");
    scanf("%f", &numin);

    na = ((numin*16)/25.4)*200;
    sm(na,nb,nc,ii);
    printf("Check stage position.  Satisfied? ('y' to proceed) ");
    scanf("%s",gostrng);

  }
  while(gostrng[0] != 'y');

/*  Define file names, enter start & end, step distance  */

  printf("Enter the step length in mm. ");
  scanf("%f", &delta);
  printf("Enter the number of cuts to take (max of 100). ");
  scanf("%d", &cuts);
  printf("Enter the starting position (in mm) ");
  scanf("%f", &startpos);
  printf("Enter a six letter file designator for this data set. ");
  scanf("%s", desig);
  numin = delta;
/* initialize filenames for every cut, numbered sequentially from 0 */
  for(i=0;i<cuts;i++)
  {
    for(j=0;j<6;j++)
      names[i][j]=desig[j];
    names[i][8]='.';
    names[i][9]='i';
    names[i][10]='r';
    names[i][11]='s';
  }
  namer = div(cuts,10);
  for(j=0;j<namer.quot;j++)
  {
    for(i=0;i<=9;i++)
```

```
    {
     names[10*j+i][6]='0'+j;
     names[10*j+i][7]='0'+i;
    }
  }
  for(i=0;i<namer.rem;i++)
  {
   names[10*namer.quot+i][6]='0'+namer.quot;
   names[10*namer.quot+i][7]='0'+i;
  }

/* initialize frame grabber board */
  istat = is_initialize();
  istat = is_configuration(conary);
  istat = is_set_sync_source(1);
  istat = is_select_input_frame(0);
  istat = is_select_output_frame(0);
  istat = is_display(1);

/* allocate memory for 17 images in PC RAM */
/* 16 for single shots plus one for the average */
  for(j=2;j<=18;j++)
  {
   istat = is_allocate(j);
  }

/* set camera gain and focus */
  printf("Set the gain and focus the camera. ('y' when ready). \n");
  istat = is_passthru();
  do
  {
    scanf("%s",gostrng);
  }
  while(gostrng[0] != 'y');
  istat = is_freeze_frame();

/*  LOOP  */
/* loop will execute once for each plane desired */

  for(i=0;i<cuts;i++)
  {

/*  Acquire 16 frames */
    nptr = &names[i][0];
    frame = 0;
    for(j=2;j<=17;j++)
    {
REDO:
/* check registers for the trigger signal to the frame grabber */
      do
      {
```

```
            regcon = _inpw(0x230);
            regout = regcon & regmask;
      }
      while(regout == 32);
/* once trigger signal is received (regout =32), acquire an image */
      istat = is_acquire(0,1);
/* image sum used to check if camera and excimer laser were in sync -- if not, retake image */
      istat = is_summation(0,&isum);
      if(isum <= minsum) goto REDO;
      istat = is_frame_copy(0, j);
/* copy acquired image into a RAM frame from the on-board memory */
/* repeat for 16 frames */
    }

/* Average the 16 frames into one, one row at a time */
    for(k=0;k<480;k++)
    {
      row0 = k;
      istat = is_set_active_region(row0, 0, 1, 512);
      for(j=2;j<=17;j++)
      {
/* get one row of one image out of RAM */
            istat = is_get_region(j, lines);
/* add each pixel of this image into an averaging array */
            for(jj=0;jj<512;jj++)
              avary[jj] += lines[jj];
/* repeat for each of the 16 images */
      }
/* divide each element of the ageraging array by the number of frames (16)
      for(jj=0;jj<512;jj++)
            avary[jj] = avary[jj] / 16;
/* put the averaged line into the last RAM frame buffer */
      istat = is_put_region(18, avary);
/* reset averaging array to zero */
      for(jj=0;jj<512;jj++)
            avary[jj] = 0;
/* repeat for the 480 lines of the image */
    }

/* copy the averaged frame to a buffer on the frame grabber card for display and saving */
    istat = is_frame_copy(18,0);
    istat = is_set_active_region(0, 0, 480, 512);
    istat = is_save(frame, enc, reso, brt, nptr);
/* clear average frame in RAM */
    istat = is_frame_clear(18);
    printf("Station #%d Completed\n",i);

/*  Move stage to next station        */

    if(i >= (cuts - 1))
      goto ENDER;
```

```
    ii = 2;
/*    nb = 0;  0 = moving stage down to lower jet */
    nb = 1;   /* 1 = moving stage "up" to lower jet */
    na = ((numin*16)/25.4)*200;
    sm(na,nb,nc,ii);

/* timer to give jet time to settle out any vibration of the stage before imaging */
    time(&timer1);
    do
    {
    time(&timer2);
    dtime = difftime(timer2, timer1);
    }
    while(dtime <= .25);

/*  END LOOP -- repeat for each cut desired */

  }

ENDER:

/* write out header file for use in data conversion program */
/* includes # of cuts, # mm between cuts, and mm position of first cut */
  istat = is_end();
  for(i=0;i<=5;i++)
  {
    hdrfile[i] = desig[i];
  }
  hdrfile[6] = '.';
  hdrfile[7] = 'h';
  hdrfile[8] = 'd';
  hdrfile[9] = 'r';
  hdrfile[10] = 0;
  fp = fopen(hdrfile, "w");
  fprintf(fp,"%d\n%f\n%f\n", cuts, delta, startpos);
  fclose(fp);

}

/* Stage mover routine */
void sm(long int na, int nb, int nc, int ii)
{
  short status, t1;
  int jjj, i, ic, n[2][4];
  long int i1, j;
  unsigned short j1, j2, j3, j4, t2, t3;


  t1 = 0;
  t2 = 255;
  t3 = 136;
```

```
  status = enable_for_output(t1);
  n[0][0]=128;
  n[0][1]=130;
  n[0][2]=132;
  n[0][3]=134;
  n[1][0]=8;
  n[1][1]=40;
  n[1][2]=72;
  n[1][3]=104;
  i1=na;
  ic=0;
  if (nc != 0) ic=1;
  j1=n[ic][0];
  j2=n[ic][1];
  j3=n[ic][2];
  j4=n[ic][3];
  if (nb != 0) goto CCW;

/* CW Direction */

  for(j=1;j<=i1;j++)
  {
    for(i=1;i<=ii;i++)
      status = output_digital_value(t1,t2,j2);
    for(i=1;i<=ii;i++)
      status = output_digital_value(t1,t2,j1);
  }
  goto SKIP;

/* CCW direction */

CCW:
  for(j=1;j<=i1;j++)
  {
    for(i=1;i<=ii;i++)
      status = output_digital_value(t1,t2,j4);
    for(i=1;i<=ii;i++)
      status = output_digital_value(t1,t2,j3);
  }

SKIP:
   for(j=1;j<=100000;j++)
    continue;
   status = output_digital_value(t1,t2,t3);
   return;
}
```

A.3 Data Reduction Code: DETAIL2.C

```
#include <stdlib.h>
#include <stdio.h>
#include <isdefs.c>
#include <math.h>
#include <time.h>

main()
{

  char inname[13], outname[13], again[5], nextname[13];
  char normname[13], bgname[13], savename[13], title[52];
  int i, j, jj, cmax, nmax, bgflag, savflag, sptot, spav, xmax, ymax;
  int xtl, ytl, xbr, ybr, convary[15], sprpxl[100], stepsq, zoom;
  int xp, yp, x, y, step, fp, cuts, xxx, yyy, doneline[512], dstep;
  int normline[512], bgline[512], pictline[512], istat, conary[20];
  long pcount;
  float xr, yr, zr, ratio, ppd, diam, brt, xrem, yrem, fmax;
  float delta, startpos, diamm, ppmm, tbrt, ctr, ntr, btr;
  div_t temp;
  double dtime;
  time_t timer1, timer2;


  FILE *fp2;

  sptot = 0;
  pcount = 0;
  cmax = 0;
  nmax = 0;
  ppmm = 2.9;
  ppd = 0.0437139;
  diam = 0.7;
  diamm = 7;
  ratio = 1.25;
  xmax = 0;
  ymax = 0;
  xp = 512;
  yp = 480;
  zoom = 1;
  for(i=0;i<=12;i++)
  {
    inname[i] = 0;
    outname[i] = 0;
  }
RESTEP:
  printf("Enter the number of pixels per millimeter: ");
  scanf("%f",&ppmm);
  printf("There are %4.2f pixels per mm. \n", ppmm);
  printf("Note: For best results, use an even number of pixels (2,4,etc.).\n");
```

```
  printf("Enter the size superpixels would you like (in pixels).\n");
  printf("\t(ex: 2 = 2x2, 1 = no binning, max = 10, integer only): ");
  scanf("%u", &step);
  if(step > 10) goto RESTEP;
  stepsq = step*step;
  printf("Enter zoom factor for data (1=as is, 2=2x): ");
  scanf("%u", &zoom);

/* Read in header file generated by the imaging program */
/* Format is # of cuts (int), # of mm between planes, mm pos of first plane */

  printf("Enter the six letter designation of the input file. \n");
  scanf("%s", inname);
  inname[6] = '.';
  inname[7] = 'h';
  inname[8] = 'd';
  inname[9] = 'r';
  inname[10] = 0;

  fp2 = fopen(inname, "r");
  fscanf(fp2,"%d %f %f", &cuts, &delta, &startpos);
  fclose(fp2);

/* Prepare file names */

  for(i=0;i<=5;i++)
  {
    nextname[i] = inname[i];
    outname[i] = inname[i];
  }
  nextname[6] = ('0');
  nextname[7] = ('0'-1);
  nextname[8] = '.';
  nextname[9] = 'i';
  nextname[10] = 'r';
  nextname[11] = 's';
  nextname[12] = 0;
  outname[6] = '.';
  outname[7] = 't';
  outname[8] = 'x';
  outname[9] = 't';
  outname[10] = 0;

  printf("Output file name will be %s. \n",outname);
  fp2 = fopen(outname, "w");
  zr = startpos;

/* IRIS Initializations */

  istat = is_initialize();
  istat = is_configuration(conary);
```

```
  istat = is_set_sync_source(0);
  istat = is_select_input_frame(0);
  istat = is_select_output_frame(0);
  istat = is_display(1);
  for(j=2;j<=5;j++)
  {
    istat = is_allocate(j);
    istat = is_frame_clear(j);
  }
  istat = is_frame_clear(0);
  istat = is_frame_clear(1);

/* Write header file for Tecplot (Preplot) */

  printf("Enter the title for the Tecplot file (50 char max, no spaces). \n");
  scanf("%s", title);
  fprintf(fp2, "TITLE = \"%s\" \n", title);
  fprintf(fp2, "VARIABLES = \"X\", \"Y\", \"Z\", \"Intensity\" \n");
  dstep = step * zoom;
  temp = div(xp,dstep);
  xxx = temp.quot;
  if(temp.rem != 0)
    xxx++;
  temp = div(yp,dstep);
  yyy = temp.quot;
  if(temp.rem != 0)
    yyy++;
  fprintf(fp2, "ZONE I=%d, J=%d, K=%d, F=POINT \n", xxx, yyy, cuts);

  printf("Would you like to use background subtraction? (0 = yes) \n");
  scanf("%d", &bgflag);
  if(bgflag != 0) goto SKIPBG;
  printf("Enter the full name of the background file. \n");
  scanf("%s", bgname);
  bgname[12] = 0;
  istat = is_restore(0,0,0,bgname);
  istat = is_frame_copy(0,3);

/* Perform binning to create superpixels for background frame */
            sum the elements of the array,
            and divide by the number of the pixels in the

  for(y=0; y < yp; y+=step)
  {
  printf("*");
    for(x = 0; x < xp; x+=step)
    {

/* read the pixels in the superpixel into an array */
      istat = is_set_active_region(y, x, step, step);
      istat = is_get_region(3, sprpxl);
```

```
/* sum the pixels in the array, skipping the lines in the even frame */
     for(jj=0;jj<step;jj+=2)
     {
       for(i=0; i<step; i++)
       sptot += sprpxl[i+step*jj];
     }
/* divide the sum by the number of pixels in the superpixel */
     spav = sptot / (stepsq/2);
/* put the average value into all elements of the array */
     for(i=0; i<stepsq; i++)
           sprpxl[i] = spav;
/* put the values from the array into the image */
     istat = is_put_region(3, sprpxl);
     sptot = 0;
   }
  }
  printf("\n");
  istat = is_frame_copy(3,0);
  time(&timer1);
  do
  {
   time(&timer2);
   dtime = difftime(timer2, timer1);
  }
  while(dtime <= 1);


SKIPBG:

  printf("Enter the full name of the normalization file. \n");
  scanf("%s", normname);
  normname[12] = 0;
  istat = is_restore(0,0,0,normname);
  istat = is_frame_copy(0,2);
  printf("Do you want to save the normalized images? (0 = yes): ");
  scanf("%d", &savflag);

/* Perform binning to create superpixels for normalization frame */

  for(y=0; y < yp; y+=step)
  {
  printf("*");
    for(x = 0; x < xp; x+=step)
    {
      istat = is_set_active_region(y, x, step, step);
      istat = is_get_region(2, sprpxl);
/* sum the pixels in the array, skipping the lines in the even frame */
      for(jj=0;jj<step;jj+=2)
      {
        for(i=0; i<step; i++)
        sptot += sprpxl[i+step*jj];
```

```
      }
/* divide the sum by the number of pixels in the superpixel */
      spav = sptot / (stepsq/2);
      for(i=0; i<stepsq; i++)
            sprpxl[i] = spav;
      istat = is_put_region(2, sprpxl);
      sptot = 0;
     }
   }
   printf("\n");
   istat = is_frame_copy(2,0);
   time(&timer1);
   do
   {
    time(&timer2);
    dtime = difftime(timer2, timer1);
   }
   while(dtime <= 1);

/* Start working with the data image */

   for(j=0;j<cuts;j++)
   {
    fmax = 0;
    cmax = 0;
    nmax = 0;
    nextname[7] += 1;
    if(nextname[7] == ('9'+1))
    {
     nextname[7] = '0';
     nextname[6] += 1;
    }

    istat = is_restore(0,0,0,nextname);
    istat = is_frame_copy(0,4);

/* Perform binning of the data image to create superpixels */

    for(y=0; y < yp; y+=step)
    {
     printf("*");
     for(x = 0; x < xp; x+=step)
     {
           istat = is_set_active_region(y, x, step, step);
           istat = is_get_region(4, sprpxl);
/* sum the pixels in the array, skipping the lines in the even frame */
            for(jj=0;jj<step;jj+=2)
            {
             for(i=0; i<step; i++)
             sptot += sprpxl[i+step*jj];
            }
```

```
/* divide the sum by the number of pixels in the superpixel */
              spav = sptot / (stepsq/2);
            for(i=0; i<stepsq; i++)
              sprpxl[i] = spav;
            istat = is_put_region(4, sprpxl);
            sptot = 0;
     }
    }
    printf("\n");

/* Check data for max value */

    for(y = 0; y < yp; y++)
    {
      istat = is_set_active_region(y, 0, 1, 512);
      istat = is_get_region(4, pictline);
      istat = is_get_region(2, normline);
      istat = is_get_region(3, bgline);

      for(x = 0; x < xp; x++)
      {
            if(normline[x] == 0)
              normline[x] = 1;

            ctr = pictline[x];
            ntr = normline[x];
            btr = bgline[x];
            tbrt = ctr - btr;
            if(tbrt < 0) tbrt = 0;
            tbrt = (tbrt / ntr) * 512;
            if(tbrt > fmax)
            {
/* Max value determination is limited to the center region of the image */
              if((x > 100) & (x < 412) & (y > 100) & (y < 380))
              {
                fmax = tbrt;
                if(j==0)
                {
                  xmax = x;
                  ymax = y;
                }
              }
            }
            if(pictline[x] > cmax)
              cmax = pictline[x];
            if(normline[x] > nmax)
              nmax = normline[x];
     }
    }
    printf("%d %5.2f  %d  %d \n",j,fmax,cmax,nmax);
```

```
/* Create normalized image */

   for(y=0; y < yp; y++)
   {
     istat = is_set_active_region(y, 0, 1, 512);
     istat = is_get_region(4, pictline);
     istat = is_get_region(2, normline);
     istat = is_get_region(3, bgline);

     for(x = 0; x < xp; x++)
     {
          if(normline[x] == 0)
            normline[x] = 1;

          ctr = pictline[x];
          btr = bgline[x];
          ntr = normline[x];
/* subtract background image from data image */
          tbrt = ctr - btr;
          if(tbrt < 0) tbrt = 0;
/* divide by normalization image and scale up */
          tbrt = (tbrt / ntr) * 512;
/* scale corrected data to a range of 0 to 1 */
          brt = tbrt / fmax;
/* scale corrected data to a range of 0 to 255 */
          doneline[x] = brt * 255;
     }
/* put scaled data into image buffer in memory */
     istat = is_put_region(5, doneline);
   }

/* Write data out in x,y,z,intensity format */

   xtl = xmax - ((512 / zoom) / 2);
   ytl = ymax - ((480 / zoom) / 2);
   xtl += step/2;
   ytl += step/2;
   xbr = xmax + ((512 / zoom) / 2);
   ybr = ymax + ((480 / zoom) / 2);

   for(y=ytl; y < ybr; y+=step)
   {
     istat = is_set_active_region(y, 0, 1, 512);
     istat = is_get_region(5,pictline);
     for(x=xtl; x < xbr; x+=step)
     {
/* transform x and y from pixels to mm */
          xr = x * ratio / ppmm;
          yr = y / ppmm;
          brt = pictline[x];
/* transform data from 0-255 to 0-1 */
```

```
            brt = brt / 255;
            pcount++;
/* write out data to text file */
            fprintf(fp2, "%5.3f %5.3f %5.3f %4.2f \n",xr,yr,zr,brt);
      }
    }

/* Save normalized image and prepare for next station */

    istat = is_frame_copy(5,0);
    if(savflag == 0)
    {
      for(jj=0;jj<13;jj++)
            savename[jj] = nextname[jj];
      savename[4] = nextname[5];
      savename[5] = nextname[6];
      savename[6] = nextname[7];
      savename[7] = 'n';

      istat = is_set_active_region(0, 0, 480, 512);
      istat = is_save(0, 0, 1, 0, savename);
    }

    time(&timer1);
    do
    {
    time(&timer2);
    dtime = difftime(timer2, timer1);
    }
    while(dtime <= 1);
    istat = is_frame_clear(0);
    istat = is_frame_clear(1);
    istat = is_frame_clear(4);
    istat = is_frame_clear(5);

    zr += delta;
  }

/* Finish Up */

  fclose(fp2);
  printf("Total points written: %lu",pcount);
  istat = is_end();
  return 0;
}
```

APPENDIX B

STANFORD COMPUTER OPTICS 4 QUIK 5 ICCD CAMERA

B.1 Description

The Stanford Computer Optics 4 Quik 5 camera is an intensified, gated, scientific grade charge-coupled display (CCD) camera. The spectral response of the camera is from 180 nm to 820 nm, allowing for imaging from the ultraviolet through the visible and into the infrared. The CCD array consists of 768 x 494 pixels, with 640 x 480 pixels active, and an image area of 6mm x 4.5mm. The intensifier is a proximity focused microchannel plate (MCP), which is lens-coupled to the CCD. The MCP operates at voltages from 0 - 1000 V, with a maximum photonic gain of greater than 10^6. The MCP also controls the shutter time, with exposures down to 5 ns possible with external control of the MCP, or 50 ns if controlled internally.

The camera can be controlled either by the internal microprocessor or through externally applied TTL pulses. An RS 232 is used to communicate with the internal microprocessor to set exposure time, delay, and intensifier voltage. The camera shutter and delay times can also be controlled externally using TTL pulses to directly control the operation of the MCP, which yields faster shutter times and less jitter than with internal control. The camera has a built in delay of 30 ns from triggering until shutter opening. The jitter in the shutter opening time is 25 ns when using internal control and is not measureable when being externally controlled.

B.2 Incident Light Intensity Warning

There are several problems inherent in the use of the 4 Quik 5 camera. The first is the inherent sensitivity of the intensifier to light. The intensifier gain must be kept at or above 600 V in order to help prevent damage to the phosphor. If the intensifier is charged to too low a voltage and a long exposure time is used to increase the intensity of the image, the phosphor can be severely damaged. For example, to take images in typical room lighting, the

gain should be set to 750 V and the gate time set to 0.00001 seconds. This is extremely critical for experiments where low intensity emissions are being collected, because if long exposure times are necessary to achieve a sufficient signal level, great care must be taken to ensure that the camera is reset to native mode (500 V gain, 50 ns exposure time) before being exposed to room light. Reflection of the light source into the camera during such experiments can also lead to damage of the phosphor, because the intensity of the light source is much greater than that of the scattered light.

B.3 Timing and Triggering

The gating of the MCP can be controlled either by the internal microprocessor or an external TTL pulse generator. In this study, the internal microprocessor was used. Using the internal microprocessor for control, commands are fed to the CPU using the RS 232 interface. The shutter time can be set over the range from 50 ns to 8 sec in 50 ns incriments, and the shutter delay time can be set from 0 to 8 sec in 50 ns increments. The triggering of the shutter can either be synchronized with the internal reading of the CCD, or it can be controlled externally with a TTL signal. In this study, the triggering was controlled externally, using an output of the D/A card controlling the actuators to phase-lock the imaging with the excitation.

In order to assure that the shutter only opened once during a video frame, a single trigger discriminator was contructed. This circuit was contructed from TTL logic circuits, with inputs from the D/A card and a pulse train produced by the camera whenever the CCD is being read. The output from the circuit was a voltage drop when the first trigger was received per frame. The voltage was held low until the end of the next camera pulse, which indicated that the CCD had been read. Then, the voltage was set high, in readiness for the next trigger.

B.4 Data Output and Capture

The image captured on the CCD chip is processed within the camera into a standard RS 170 video signal, for use with a VCR or frame grabber. The camera was operated in frame mode to ensure correct vertical position of each image, acquire full resolution images, and for compatability with a frame grabber card. However, problems were encountered with acquiring full resolution images on the PC using the DT 2851 Frame Grabber. When the system was being operated in free-running mode, with constant image acquisition and frame grabbing, full resolution images were acquired. However, when the camera and frame grabber cards were being triggered together to acquire phase-locked data, acquisition of every second line, the even field, was uncertain. Various adjustments to the triming and triggering were attempted, but none corrected the problem. As a temporary solution, the data in the even field was discarded, and only the data collected in the reliable odd field processed. Late in the study, it was learned that the DT 2851 was not designed to handle frame transfer data,

only field transfer data. This is why the transfer of the second field from the camera to the frame grabber was uncertain. Due to the time and expense required to acquire a frame grabber card capable of proper image transfer with this camera, the experiments were continued with the reduced resolution offered by the available equipment.

APPENDIX C

LUMONICS 861-T EXCIMER LASER

C.1 Description

The Lumonics 861-T is a rare gas halide excimer laser in the Lumonics TE-860-4 Series, which generates intense pulses of radiation by transitions from excited molecules in a high pressure electrical discharge. The wavelengths available are from the ultra-violet into the visible, depending on the gas mix used. The laser is packaged in a single housing, which contains the laser head, power supply, discharge components, gas handling system, and all operating controls. With the installed electrode and capacitors, the laser was capable of output from 193 nm, with ArF, to 428 nm, with N_2^+. Using XeCl, the gas mix used in this study for output at 308 nm, the maximum pulse energy was 200 mj/pulse, a maximum average power of 20W, and a maximum pulse rate of 500 pps.

C.2 Gas Handling System

The mixture chosen for this study was Xenon Chloride, XeCl. This was chosen because it is less hazardous than the other mixture which lases in the absorption band of acetone, KrF, though both halogens are caustic and toxic. The wavelength emitted by the XeCl is also in the proper range to excite OH radicals for combustion diagnostics, and was thus judged more useful for the lab.

Due to the hazardous nature of the Hydrogen Chloride gas (HCl) needed for the XeCl gas mix, a ventilated gas cabinet was procured for storage of the HCl cylinder. Valves were acquired to apply and release pressure from outside the cabinet for several pneumatically actuated valves inside the cabinet. Thus, the cabinet would not have to be opened during the filling of the laser, minimizing the hazard to lab personnel. The cabinet was also equipped with a nitrogen purge system to flush the HCl from the lines in the cabinet when not in use.

The procedure for the use of the gas cabinet to deliver HCl to the laser follows in Attachment 1. This procedure was developed to minimize the potential for exposure of lab personnel to the HCl gas. It was also designed to deliver the HCl to the laser without fear of contamination from room air.

Before the laser can be used for a gas mix containing HCl after exposure to the atmosphere or use with other gases, the laser cavity has to be passivated. To do this, first the laser is evacuated, refilled to 16 psia with helium and evacuated again to a few torr. Then, with the AC power on and cooling water off, the laser should be filled to 16 psia with a 3% mixture of HCl with a balance of helium. This mixture is left in the cavity for two hours, and then the procedure is repeated. The laser cannot be fired with this mixture, because it will lead to severe arcing which can damage the energy storage capacitor. The laser is then filled with the proper lasing gas mixture, the cooling water turned on, and the laser fired. The procedure for filling the excimer laser with the required gas mixture follows in Attachment 2. This procedure was derived from the manual for the excimer laser, which described the generic fill procedure for the laser. The halogen injection procedures are also included in Attachment 2. The laser cavity continues to passivate during use, depleting the chloride in the gas mix. The pulse power can be partially restored and the life of the gas mixture extended by adding small amounts of HCl to the existing gas mix as the pulse power drops off. Both the pulse power and gas mixture lifetimes will increase as a result of continual use of the laser.

C.3 Operation

The laser can be run in two different modes, free running and triggered. In free running mode, the laser pulses at the rate dialed in on the controls at the rear of the laser. In triggered mode, a TTL pulse is applied to the trigger input to fire the laser in synch with an outside source.

In either mode, there are two choices for charging the thyratron, depending on the pulse rate selected. For operation on the .2x or 2x scales, .2 Hz to 20 Hz, "charge on demand" mode is selected. In this case, the laser pulse is delayed by 15 ms to allow the thyratron to charge before each pulse. For operation on the 20x scale, greater than 20 Hz, the thyratron is constantly charged, so the laser pulse follows the trigger pulse without a delay. Because the thyratron will fire randomly if left fully charged, the "charge on demand" mode is selected for low pulse rates.

Special care must be taken during the operation of the excimer laser, due to the high power and the ultraviolet radiation. All optics, and preferably the entire beam path, should be enclosed to prevent scatter off the optics from reaching personnel. The path should also end at a non-reflecting beam stop which is capable of absorbing the remaining power in the beam. All personnel in the area should still wear eye protection and as little skin should be exposed as possible to prevent injury from inadvertant exposure to the laser radiation.

APPENDIX C, ATTACHMENT 1

GAS CABINET PROCEDURE

Part I:

Verify air flow in cabinet stack
Open N2 tank, set regulator to 90 psig
Open HCl tank, set regulator to 65 psig
Close cabinet

Open valve D (N2 flush), Close valve D [pressurize laser lines]
Open valve E (vent), Close valve E [vent laser lines]
Open valve C [fill laser lines with HCl]
Open valve E, Close valve E [clear HCl line of contaminants]

FILL LASER PER FILL PROCEDURE

Part II:

Close valve C
Open cabinet
Close HCl tank
Close cabinet
Open valve C

Open valve B [vent HCl in lines to hood]
Open valve D [N2 flush of cabinet]
Close valve C [isolate cabinet from delivery lines]
Open valve E, Close valve E [N2 flush of delivery lines]
Close valve D
Open valve A [vacuum flush cabinet]
Close valve A
Close valve B

Open cabinet
Close N2 tank
Open valve A, Close valve A [vent remaining N2]

APPENDIX C, ATTACHMENT 2

LASER FILL PROCEDURE

Carry out first part of gas cabinet procedure
Set Helium regulator to 45 psig
Pump down laser to a few torr, close laser exhaust valve
Empty several calibrated volumes (HCl) into laser
Fill laser to 700 torr with Helium
Pump down laser to a few torr, close laser exhaust valve
Repeat fill and pump down one more time
Verify laser exhaust valve is closed

Set Neon regulator to 45 psig
Set Xenon regulator to 10 psig
(HCl should already be set to 65 psig)

Fill laser to 40 torr with Xenon
Fill laser to 90 torr with HCl (total pressure, partial pressure HCl in mix, 50 torr).
Use as many calibrated volumes as necessary

Close vacuum gauge isolation valve
Fill laser to 45 psig with Neon, use high pressure gauge
Close laser valves to horizontal position
Close all gas bottles (lines stay pressurized on all but HCl)
Carry out second part of gas cabinet procedure

HALOGEN INJECTION

Carry out first part of gas cabinet procedure
Inject one or two calibrated volumes of HCl into filled laser
Carry out second part of gas cabinet procedure