Picture System 2
Graphics Subroutine Package

UNIX™ Edition

September 1980

(minor revisions March 1982)

Computer Graphics Laboratory
School of Pharmacy
University of California
San Francisco, CA 94143

PERMUTED INDEX

pscopy: hardcopy generator for Picture System	2.	pscopy(1G)
psinit: initialize the Picture System	2.	psinit(3G)
pserrs: expand cryptic Picture System	2 error messages.	pserrs(1G)
psfini: close all open Picture System	2 files.	psfini(3G)
intro: introduction to Picture System	2 Graphics Subroutine Package.	intro(3G)
psreset: reset the Picture System	2 in time of crisis.	psreset(1G)
getchr: read from Picture System	2 keyboard.	getchr(3G)
drawps: draw an object stored in Picture System	2 memory.	drawps(3G)
dist3: distance between two	3-dimensional points.	dist3(3GU)
vv3, vxv3: dot and cross product of two	3-dimensional vectors.	vv3(3GU)
dotat: draw a dot at an	absolute coordinate.	dotat(3G)
lineto: draw a line in	absolute space.	lineto(3G)
moveto: move in	absolute space.	moveto(3G)
siasync: synchronize stereo image	alternator.	siasync(1G)
sia, left, right: stereo image	alternator routines.	sia(3G)
analog: read the value of an	analog channel.	analog(3G)
	analog: read the value of an analog channel.	analog(3G)
cossin: compute cosine and sine for an	angle.	cossin(3G)
	angle, dihed: calculate angles.	angle(3GU)
angle, dihed: calculate	angles.	angle(3GU)
operation.	apndps: re-initiate makeps/maksps mode of	apndps(3G)
nargs:	argument count.	nargs(3G)
matrix manipulations.	bldcon: perform transformation operations and	bldcon(3G)
manipulations.	bldps: perform transformation matrix	bldps(3G)
blink: make all subsequent lines	blink.	blink(3G)
	blink: make all subsequent lines blink.	blink(3G)
psbuf: set refresh	buffer mode.	psbuf(3G)
dowrbuf, nowrbuf:	buffer text output.	dowrbuf(3G)
rot:	build a rotation matrix.	rot(3G)
getrot, setrot:	build a rotation transformation.	getrot(3G)
scale:	build a scaling matrix.	scale(3G)
getscl, setscl:	build a scaling transformation.	getscl(3G)
tran:	build a translation matrix.	tran(3G)
gettrn, settrn:	build a translation transformation.	gettrn(3G)
angle, dihed:	calculate angles.	angle(3GU)
ispchd: is pen	changed?.	ispchd(3G)
analog: read the value of an analog	channel.	analog(3G)
cload:	character generator support.	cload(3G)
chargn: define a PS2	character set.	chargn(1G)
charsz: update	character size, font and orientation.	charsz(3G)
	chargn: define a PS2 character set.	chargn(1G)
orientation.	charsz: update character size, font and	charsz(3G)
	cload: character generator support.	cload(3G)
psfini:	close all open Picture System 2 files.	psfini(3G)
huesat: specify	color and saturation.	huesat(3G)
cossin:	compute cosine and sine for an angle.	cossin(3G)
dot: draw a dot at a relative	coordinate.	dot(3G)
dotat: draw a dot at an absolute	coordinate.	dotat(3G)
draw2d: draw two-dimensional	coordinate data.	draw2d(3G)
draw3d: draw three-dimensional	coordinate data.	draw3d(3G)
draw4d: draw homogeneous	coordinate data.	draw4d(3G)
rdtc: read transformed	coordinate data.	rdtc(3G)
matrices. errcheck:	correct for roundoff errors in rotation	errcheck(3GU)
cossin: compute	cosine and sine for an angle.	cossin(3G)
	cossin: compute cosine and sine for an angle.	cossin(3G)
nargs: argument	count.	nargs(3G)
makeob:	create a Linear Display List.	makeob(3G)
makeps, maksps:	create a Picture Memory display list.	makeps(3G)
psreset: reset the Picture System 2 in time of	crisis.	psreset(1G)
vv3, vxv3: dot and	cross product of two 3-dimensional vectors.	vv3(3GU)
pserrs: expand	cryptic Picture System 2 error messages.	pserrs(1G)

cursor: display a	cursor.	cursor(3G)
	cursor: display a cursor.	cursor(3G)
tablet: retrieve data tablet	cursor position.	tablet(3G)
draw2d: draw two-dimensional coordinate	data.	draw2d(3G)
draw3d: draw three-dimensional coordinate	data.	draw3d(3G)
draw4d: draw homogeneous coordinate	data.	draw4d(3G)
nufram: display new frame	data.	nufram(3G)
rdtc: read transformed coordinate	data.	rdtc(3G)
tablet: retrieve	data tablet cursor position.	tablet(3G)
chargn:	define a PS2 character set.	chargn(1G)
joystick: simulated interactive joystick	device.	joystick(3GU)
angle,	dihed: calculate angles.	angle(3GU)
scopes: select Picture System	Display.	scopes(3G)
cursor:	display a cursor.	cursor(3G)
drawob: output a Linear	Display List.	drawob(3G)
makeob: create a Linear	Display List.	makeob(3G)
makeps, maksps: create a Picture Memory	display list.	makeps(3G)
stopob: terminate a Linear	Display List.	stopob(3G)
stopps: terminate a Picture Memory	Display List.	stopps(3G)
hittag, hitset, hitclr:	display list hit testing.	hittag(3G)
subps:	display list structuring.	subps(3G)
rsetps: reset Picture Memory	Display Lists.	rsetps(3G)
setps: initialize for Picture Memory	Display Lists.	setps(3G)
nufram:	display new frame data.	nufram(3G)
text:	display text.	text(3G)
points.	dist3: distance between two 3-dimensional	dist3(3GU)
dist3:	distance between two 3-dimensional points.	dist3(3GU)
vectors. vv3, vxv3:	dot and cross product of two 3-dimensional	vv3(3GU)
dot: draw a	dot at a relative coordinate.	dot(3G)
dotat: draw a	dot at an absolute coordinate.	dotat(3G)
	dot: draw a dot at a relative coordinate.	dot(3G)
	dotat: draw a dot at an absolute coordinate.	dotat(3G)
	dowrbuf, nowrbuf: buffer text output.	dowrbuf(3G)
dot:	draw a dot at a relative coordinate.	dot(3G)
dotat:	draw a dot at an absolute coordinate.	dotat(3G)
lineto:	draw a line in absolute space.	lineto(3G)
line:	draw a line in relative space.	line(3G)
memory. drawps:	draw an object stored in Picture System 2	drawps(3G)
draw4d:	draw homogeneous coordinate data.	draw4d(3G)
draw3d:	draw three-dimensional coordinate data.	draw3d(3G)
draw2d:	draw two-dimensional coordinate data.	draw2d(3G)
	draw2d: draw two-dimensional coordinate data.	draw2d(3G)
	draw3d: draw three-dimensional coordinate data.	draw3d(3G)
	draw4d: draw homogeneous coordinate data.	draw4d(3G)
speed: set the Line Generator	drawing speed.	speed(3G)
	drawob: output a Linear Display List.	drawob(3G)
2 memory.	drawps: draw an object stored in Picture System	drawps(3G)
rotation matrices.	errcheck: correct for roundoff errors in	errcheck(3GU)
pserrs: expand cryptic Picture System 2	error messages.	pserrs(1G)
xerrors: expanded format for	error printout.	xerrors(3G)
errcheck: correct for roundoff	errors in rotation matrices.	errcheck(3GU)
pserrs:	expand cryptic Picture System 2 error messages.	pserrs(1G)
xerrors:	expanded format for error printout.	xerrors(3G)
pgspsh, pgspop, pgsrd: manipulate	file stack.	pgstack(3GU)
psfini: close all open Picture System 2	files.	psfini(3G)
charsz: update character size,	font and orientation.	charsz(3G)
xerrors: expanded	format for error printout.	xerrors(3G)
nufram: display new	frame data.	nufram(3G)
	fswitch: read Function Switches.	fswitch(3G)
System objects.	fulsub: subroutine to output segmented Picture	fulsub(3GU)
fswitch: read	Function Switches.	fswitch(3G)
lights: set lights on	Function Switches.	lights(3G)
setlit: set lights on	Function Switches.	setlit(3G)
inst:	generate instancing transformations.	inst(3G)
master:	generate master transformations.	master(3G)
speed: set the Line	Generator drawing speed.	speed(3G)
pscopy: hardcopy	generator for Picture System 2.	pscopy(1G)
cload: character	generator support.	cload(3G)
	getchr: read from Picture System 2 keyboard.	getchr(3G)
	getknob: get values of interactive knobs.	getknob(3GU)
transformation.	getrot, setrot: build a rotation	getrot(3G)
	getscl, setscl: build a scaling transformation.	getscl(3G)

transformation.	gettrn, settrn: build a translation	gettrn(3G)
intro: introduction to Picture System 2	Graphics Subroutine Package.	intro(3G)
pscopy:	hardcopy generator for Picture System 2.	pscopy(1G)
hitest:	hit testing.	hitest(3G)
hittag, hitset, hitclr: display list	hit testing.	hittag(3G)
hitwin: specify a	hit window.	hitwin(3G)
hittag, hitset,	hitclr: display list hit testing.	hittag(3G)
	hitest: hit testing.	hitest(3G)
hittag,	hitset, hitclr: display list hit testing.	hittag(3G)
testing.	hittag, hitset, hitclr: display list hit	hittag(3G)
	hitwin: specify a hit window.	hitwin(3G)
draw4d: draw	homogeneous coordinate data.	draw4d(3G)
	huesat: specify color and saturation.	huesat(3G)
siasync: synchronize stereo	image alternator.	siasync(1G)
sia, left, right: stereo	image alternator routines.	sia(3G)
setps:	initialize for Picture Memory Display Lists.	setps(3G)
psinit:	initialize the Picture System 2.	psinit(3G)
	inst: generate instancing transformations.	inst(3G)
inst: generate	instancing transformations.	inst(3G)
joystick: simulated	interactive joystick device.	joystick(3GU)
getknob: get values of	interactive knobs.	getknob(3GU)
Graphics Subroutine Package.	intro: introduction to Picture System 2	intro(3G)
Subroutine Package. intro:	introduction to Picture System 2 Graphics	intro(3G)
	ispchd: is pen changed?.	ispchd(3G)
	iswset: is switch set?.	iswset(3G)
joystick: simulated interactive	joystick device.	joystick(3GU)
device.	joystick: simulated interactive joystick	joystick(3GU)
jumpps:	jump to another Picture Memory object.	jumpps(3G)
	jumpps: jump to another Picture Memory object.	jumpps(3G)
getchr: read from Picture System 2	keyboard.	getchr(3G)
getknob: get values of interactive	knobs.	getknob(3GU)
sia,	left, right: stereo image alternator routines.	sia(3G)
lights: set	lights on Function Switches.	lights(3G)
setlit: set	lights on Function Switches.	setlit(3G)
	lights: set lights on Function Switches.	lights(3G)
	line: draw a line in relative space.	line(3G)
speed: set the	Line Generator drawing speed.	speed(3G)
lineto: draw a	line in absolute space.	lineto(3G)
line: draw a	line in relative space.	line(3G)
txture: set	line texture.	txture(3G)
drawob: output a	Linear Display List.	drawob(3G)
makeob: create a	Linear Display List.	makeob(3G)
stopob: terminate a	Linear Display List.	stopob(3G)
blink: make all subsequent	lines blink.	blink(3G)
	lineto: draw a line in absolute space.	lineto(3G)
drawob: output a Linear Display	List.	drawob(3G)
makeob: create a Linear Display	List.	makeob(3G)
makeps, maksps: create a Picture Memory display	list.	makeps(3G)
stopob: terminate a Linear Display	List.	stopob(3G)
stopps: terminate a Picture Memory Display	List.	stopps(3G)
hittag, hitset, hitclr: display	list hit testing.	hittag(3G)
subps: display	list structuring.	subps(3G)
rsetps: reset Picture Memory Display	Lists.	rsetps(3G)
setps: initialize for Picture Memory Display	Lists.	setps(3G)
unit:	load a unit matrix into the Picture System MAP.	unit(3GU)
lookat: produce	lookat operators.	lookat(3GU)
	lookat: produce lookat operators.	lookat(3GU)
	makeob: create a Linear Display List.	makeob(3G)
list.	makeps, maksps: create a Picture Memory display	makeps(3G)
apndps: re-initiate	makeps/maksps mode of operation.	apndps(3G)
makeps,	maksps: create a Picture Memory display list.	makeps(3G)
mmu: Memory	Management Unit.	mmu(3G)
pgspsh, pgspop, pgsrd:	manipulate file stack.	pgstack(3GU)
perform transformation operations and matrix	manipulations. bldcon:	bldcon(3G)
bldps: perform transformation matrix	manipulations.	bldps(3G)
load a unit matrix into the Picture System	MAP. unit:	unit(3GU)
	master: generate master transformations.	master(3G)
master: generate	master transformations.	master(3G)
correct for roundoff errors in rotation	matrices. errcheck:	errcheck(3GU)
rot: build a rotation	matrix.	rot(3G)
scale: build a scaling	matrix.	scale(3G)
tran: build a translation	matrix.	tran(3G)

trpose: transpose of a Picture System matrix. . . . . transpose, transpose(3GU)
unit: load a unit matrix into the Picture System MAP. . . . . unit(3GU)
bldcon: perform transformation operations and matrix manipulations. . . . . bldcon(3G)
bldps: perform transformation matrix manipulations. . . . . bldps(3G)
pop: pop the matrix stack. . . . . pop(3G)
push: push the matrix stack. . . . . push(3G)
draw an object stored in Picture System 2 memory. . . . . drawps: drawps(3G)
wbtmem, rbtmem: write back to memory. . . . . wbtmem(3G)
makeps, maksps: create a Picture Memory display list. . . . . makeps(3G)
stopps: terminate a Picture Memory Display List. . . . . stopps(3G)
rsetps: reset Picture Memory Display Lists. . . . . rsetps(3G)
setps: initialize for Picture Memory Display Lists. . . . . setps(3G)
mmu: Memory Management Unit. . . . . mmu(3G)
jumpps: jump to another Picture Memory object. . . . . jumpps(3G)
menuset, menucheck, menu_box: Picture System menu package. . . . . menu(3GU)
menuset, menucheck, menu_box: Picture System menu package. . . . . menu(3GU)
menucheck, menu_box: Picture System menu . . . . . menu(3GU)
menuset, menucheck, menu_box: Picture System . . . . . menu(3GU)
pserrs: expand cryptic Picture System 2 error messages. . . . . pserrs(1G)
mmu: Memory Management Unit. . . . . mmu(3G)
psbuf: set refresh buffer mode. . . . . psbuf(3G)
stopwb: terminate write back mode. . . . . stopwb(3G)
apndps: re-initiate makeps/maksps mode of operation. . . . . apndps(3G)
moveto: move in absolute space. . . . . moveto(3G)
move: move in relative space. . . . . move(3G)
move: move in relative space. . . . . move(3G)
moveto: move in absolute space. . . . . moveto(3G)
nargs: argument count. . . . . nargs(3G)
dowrbuf, nowrbuf: buffer text output. . . . . dowrbuf(3G)
nufram: display new frame data. . . . . nufram(3G)
jumpps: jump to another Picture Memory object. . . . . jumpps(3G)
drawps: draw an object stored in Picture System 2 memory. . . . . drawps(3G)
subroutine to output segmented Picture System objects. . . . . fulsub: fulsub(3GU)
psfini: close all open Picture System 2 files. . . . . psfini(3G)
apndps: re-initiate makeps/maksps mode of operation. . . . . apndps(3G)
bldcon: perform transformation operations and matrix manipulations. . . . . bldcon(3G)
lookat: produce lookat operators. . . . . lookat(3GU)
charsz: update character size, font and orientation. . . . . charsz(3G)
dowrbuf, nowrbuf: buffer text output. . . . . dowrbuf(3G)
drawob: output a Linear Display List. . . . . drawob(3G)
fulsub: subroutine to output segmented Picture System objects. . . . . fulsub(3GU)
to Picture System 2 Graphics Subroutine Package. . . . . intro: introduction intro(3G)
menucheck, menu_box: Picture System menu package. . . . . menuset, menu(3GU)
ispchd: is pen changed?. . . . . ispchd(3G)
bldps: perform transformation matrix manipulations. . . . . bldps(3G)
manipulations. . . . . bldcon: perform transformation operations and matrix . . . . . bldcon(3G)
pgspsh, pgspop, pgsrd: manipulate file stack. . . . . pgstack(3GU)
pgspsh, pgspop, pgsrd: manipulate file stack. . . . . pgstack(3GU)
pgspsh, pgspop, pgsrd: manipulate file stack. . . . . pgstack(3GU)
makeps, maksps: create a Picture Memory display list. . . . . makeps(3G)
stopps: terminate a Picture Memory Display List. . . . . stopps(3G)
rsetps: reset Picture Memory Display Lists. . . . . rsetps(3G)
setps: initialize for Picture Memory Display Lists. . . . . setps(3G)
jumpps: jump to another Picture Memory object. . . . . jumpps(3G)
pscopy: hardcopy generator for Picture System 2. . . . . pscopy(1G)
psinit: initialize the Picture System 2. . . . . psinit(3G)
pserrs: expand cryptic Picture System 2 error messages. . . . . pserrs(1G)
psfini: close all open Picture System 2 files. . . . . psfini(3G)
intro: introduction to Picture System 2 Graphics Subroutine Package. . . . . intro(3G)
psreset: reset the Picture System 2 in time of crisis. . . . . psreset(1G)
getchr: read from Picture System 2 keyboard. . . . . getchr(3G)
drawps: draw an object stored in Picture System 2 memory. . . . . drawps(3G)
scopes: select Picture System Display. . . . . scopes(3G)
unit: load a unit matrix into the Picture System MAP. . . . . unit(3GU)
transpose, trpose: transpose of a Picture System matrix. . . . . transpose(3GU)
menuset, menucheck, menu_box: Picture System menu package. . . . . menu(3GU)
fulsub: subroutine to output segmented Picture System objects. . . . . fulsub(3GU)
psstat: report Picture System statistics. . . . . psstat(1G)
dist3: distance between two 3-dimensional points. . . . . dist3(3GU)
pop: pop the matrix stack. . . . . pop(3G)
pop: pop the matrix stack. . . . . pop(3G)
tablet: retrieve data tablet cursor position. . . . . tablet(3G)

	pot: a simulated potentiometer.	pot(3GU)
pot: a simulated	potentiometer.	pot(3GU)
xerrors: expanded format for error	printout.	xerrors(3G)
lookat:	produce lookat operators.	lookat(3GU)
vv3, vxv3: dot and cross	product of two 3-dimensional vectors.	vv3(3GU)
chargn: define a	PS2 character set.	chargn(1G)
	psbuf: set refresh buffer mode.	psbuf(3G)
2.	pscopy: hardcopy generator for Picture System	pscopy(1G)
messages.	pserrs: expand cryptic Picture System 2 error	pserrs(1G)
	psfini: close all open Picture System 2 files.	psfini(3G)
	psinit: initialize the Picture System 2.	psinit(3G)
crisis.	psreset: reset the Picture System 2 in time of	psreset(1G)
	psstat: report Picture System statistics.	psstat(1G)
	push: push the matrix stack.	push(3G)
push:	push the matrix stack.	push(3G)
wbtmem,	rbtmem: write back to memory.	wbtmem(3G)
	rdtc: read transformed coordinate data.	rdtc(3G)
getchr:	read from Picture System 2 keyboard.	getchr(3G)
fswitch:	read Function Switches.	fswitch(3G)
analog:	read the value of an analog channel.	analog(3G)
rdtc:	read transformed coordinate data.	rdtc(3G)
psbuf: set	refresh buffer mode.	psbuf(3G)
apndps:	re-initiate makeps/maksps mode of operation.	apndps(3G)
dot: draw a dot at a	relative coordinate.	dot(3G)
line: draw a line in	relative space.	line(3G)
move: move in	relative space.	move(3G)
psstat:	report Picture System statistics.	psstat(1G)
rsetps:	reset Picture Memory Display Lists.	rsetps(3G)
psreset:	reset the Picture System 2 in time of crisis.	psreset(1G)
tablet:	retrieve data tablet cursor position.	tablet(3G)
sia, left,	right: stereo image alternator routines.	sia(3G)
	rot: build a rotation matrix.	rot(3G)
errcheck: correct for roundoff errors in	rotation matrices.	errcheck(3GU)
rot: build a	rotation matrix.	rot(3G)
getrot, setrot: build a	rotation transformation.	getrot(3G)
errcheck: correct for	roundoff errors in rotation matrices.	errcheck(3GU)
sia, left, right: stereo image alternator	routines.	sia(3G)
	rsetps: reset Picture Memory Display Lists.	rsetps(3G)
huesat: specify color and	saturation.	huesat(3G)
	scale: build a scaling matrix.	scale(3G)
scale: build a	scaling matrix.	scale(3G)
getscl, setscl: build a	scaling transformation.	getscl(3G)
	scopes: select Picture System Display.	scopes(3G)
vwport: set	screen viewport.	vwport(3G)
fulsub: subroutine to output	segmented Picture System objects.	fulsub(3GU)
scopes:	select Picture System Display.	scopes(3G)
chargn: define a PS2 character	set.	chargn(1G)
iswset: is switch	set?.	iswset(3G)
lights:	set lights on Function Switches.	lights(3G)
setlit:	set lights on Function Switches.	setlit(3G)
txture:	set line texture.	txture(3G)
psbuf:	set refresh buffer mode.	psbuf(3G)
vwport:	set screen viewport.	vwport(3G)
speed:	set the Line Generator drawing speed.	speed(3G)
window:	set window.	window(3G)
	setlit: set lights on Function Switches.	setlit(3G)
Lists.	setps: initialize for Picture Memory Display	setps(3G)
getrot,	setrot: build a rotation transformation.	getrot(3G)
getscl,	setscl: build a scaling transformation.	getscl(3G)
gettrn,	settrn: build a translation transformation.	gettrn(3G)
routines.	sia, left, right: stereo image alternator	sia(3G)
	siasync: synchronize stereo image alternator.	siasync(1G)
joystick:	simulated interactive joystick device.	joystick(3GU)
pot: a	simulated potentiometer.	pot(3GU)
cossin: compute cosine and	sine for an angle.	cossin(3G)
charsz: update character	size, font and orientation.	charsz(3G)
line: draw a line in relative	space.	line(3G)
lineto: draw a line in absolute	space.	lineto(3G)
move: move in relative	space.	move(3G)
moveto: move in absolute	space.	moveto(3G)
hitwin:	specify a hit window.	hitwin(3G)
huesat:	specify color and saturation.	huesat(3G)

speed: set the Line Generator drawing — speed. . . . speed(3G)
speed: set the Line Generator drawing speed. . . . speed(3G)
pgspsh, pgspop, pgsrd: manipulate file — stack. . . . pgstack(3GU)
pop: pop the matrix — stack. . . . pop(3G)
push: push the matrix — stack. . . . push(3G)
psstat: report Picture System — statistics. . . . psstat(1G)
siasync: synchronize — stereo image alternator. . . . siasync(1G)
sia, left, right: — stereo image alternator routines. . . . sia(3G)
stopob: terminate a Linear Display List. . . . stopob(3G)
List. . . . — stopps: terminate a Picture Memory Display . . . . stopps(3G)
stopwb: terminate write back mode. . . . stopwb(3G)
drawps: draw an object — stored in Picture System 2 memory. . . . drawps(3G)
subps: display list — structuring. . . . subps(3G)
subps: display list structuring. . . . subps(3G)
introduction to Picture System 2 Graphics — Subroutine Package. . . . intro: intro(3G)
objects. . . . fulsub: — subroutine to output segmented Picture System . . . . fulsub(3GU)
blink: make all — subsequent lines blink. . . . blink(3G)
cload: character generator — support. . . . cload(3G)
iswset: is — switch set?. . . . iswset(3G)
fswitch: read Function — Switches. . . . fswitch(3G)
lights: set lights on Function — Switches. . . . lights(3G)
setlit: set lights on Function — Switches. . . . setlit(3G)
siasync: — synchronize stereo image alternator. . . . siasync(1G)
pscopy: hardcopy generator for Picture — System 2. . . . pscopy(1G)
psinit: initialize the Picture — System 2. . . . psinit(3G)
pserrs: expand cryptic Picture — System 2 error messages. . . . pserrs(1G)
psfini: close all open Picture — System 2 files. . . . psfini(3G)
intro: introduction to Picture — System 2 Graphics Subroutine Package. . . . intro(3G)
psreset: reset the Picture — System 2 in time of crisis. . . . psreset(1G)
getchr: read from Picture — System 2 keyboard. . . . getchr(3G)
drawps: draw an object stored in Picture — System 2 memory. . . . drawps(3G)
scopes: select Picture — System Display. . . . scopes(3G)
unit: load a unit matrix into the Picture — System MAP. . . . unit(3GU)
transpose, trpose: transpose of a Picture — System matrix. . . . transpose(3GU)
menuset, menucheck, menu_box: Picture — System menu package. . . . menu(3GU)
fulsub: subroutine to output segmented Picture — System objects. . . . fulsub(3GU)
psstat: report Picture — System statistics. . . . psstat(1G)
tablet: retrieve data — tablet cursor position. . . . tablet(3G)
tablet: retrieve data tablet cursor position. . . . tablet(3G)
stopob: — terminate a Linear Display List. . . . stopob(3G)
stopps: — terminate a Picture Memory Display List. . . . stopps(3G)
stopwb: — terminate write back mode. . . . stopwb(3G)
hitest: hit — testing. . . . hitest(3G)
hittag, hitset, hitclr: display list hit — testing. . . . hittag(3G)
text: display — text. . . . text(3G)
text: display text. . . . text(3G)
dowrbuf, nowrbuf: buffer — text output. . . . dowrbuf(3G)
txture: set line — texture. . . . txture(3G)
draw3d: draw — three-dimensional coordinate data. . . . draw3d(3G)
tran: build a translation matrix. . . . tran(3G)
getrot, setrot: build a rotation — transformation. . . . getrot(3G)
getscl, setscl: build a scaling — transformation. . . . getscl(3G)
gettrn, settrn: build a translation — transformation. . . . gettrn(3G)
bldps: perform — transformation matrix manipulations. . . . bldps(3G)
manipulations. . . . bldcon: perform — transformation operations and matrix . . . . bldcon(3G)
inst: generate instancing — transformations. . . . inst(3G)
master: generate master — transformations. . . . master(3G)
rdtc: read — transformed coordinate data. . . . rdtc(3G)
tran: build a — translation matrix. . . . tran(3G)
gettrn, settrn: build a — translation transformation. . . . gettrn(3G)
transpose, trpose: — transpose of a Picture System matrix. . . . transpose(3GU)
System matrix. . . . — transpose, trpose: transpose of a Picture . . . . transpose(3GU)
transpose, — trpose: transpose of a Picture System matrix. . . . transpose(3GU)
draw2d: draw — two-dimensional coordinate data. . . . draw2d(3G)
txture: set line texture. . . . txture(3G)
mmu: Memory Management — Unit. . . . mmu(3G)
System MAP. . . . — unit: load a unit matrix into the Picture . . . . unit(3GU)
unit: load a — unit matrix into the Picture System MAP. . . . unit(3GU)
charsz: — update character size, font and orientation. . . . charsz(3G)
analog: read the — value of an analog channel. . . . analog(3G)
getknob: get — values of interactive knobs. . . . getknob(3GU)
dot and cross product of two 3-dimensional — vectors. . . . vv3, vxv3: vv3(3GU)

vwport: set screen viewport. . . . . vwport(3G)
3-dimensional vectors. . . . . vv3, vxv3: dot and cross product of two . . . . . vv3(3GU)
vwport: set screen viewport. . . . . vwport(3G)
3-dimensional vectors. . . . . vv3, vxv3: dot and cross product of two . . . . . vv3(3GU)
wbtmem, rbtmem: write back to memory. . . . . wbtmem(3G)
hitwin: specify a hit window. . . . . hitwin(3G)
window: set window. . . . . window(3G)
window: set window. . . . . window(3G)
stopwb: terminate write back mode. . . . . stopwb(3G)
wbtmem, rbtmem: write back to memory. . . . . wbtmem(3G)
xerrors: expanded format for error printout. . . . . xerrors(3G)

NAME

intro – introduction to Picture System 2 Graphics Subroutine Package

SYNOPSIS

#include <ps.h>

DESCRIPTION

This sub-section describes procedures found in the Picture System 2 Graphics Subroutine Package (PS2GSP) and the Graphics Utility Library. Procedures are divided into various libraries dependent upon both the implementation language of the calling procedure and the section number at the top of the page:

(3G) These procedure form the basic interface to the Picture System 2 and allow the user access to all interactive devices and the full hardware capabilities of the Picture Processor. The procedures are available in two libraries, *libg* and *libgf*. The link editor *ld(1)* searches the C version of the library under the '-lg' option and the f77 version under '-lgf'. Macro, constant and typedef declarations may be obtained from the <ps.h> include file. These procedures closely match those available from Evans & Sutherland for DEC operating systems.

(3GU) These procedures implement many of the commonly needed advanced functions found in graphics programs. Examples include pseudo-potentiometers and joysticks, 'lightpen' menus and geometric operations. The C version of the library is *libgu* and the f77 version is *libguf* (*ld* options '-lgu' and '-lguf' respectively). Many of these procedures also require functions from the UNIX math library, *libm*. This library is searched automatically by the Fortran compiler *f77(1)*, but must be invoked with the '-lm' option from C compiler *cc(1)*.

DIAGNOSTICS

Error detection is performed by all of the PS2GSP procedures to ensure program integrity and to facilitate program debugging. Errors are reported by a message on the standard error output, *stderr*, in the format:

Error x detected in graphics subroutine y

followed by a call to the UNIX *abort(3)* subroutine in order to produce a core image for debugging purposes. In this case *x* is the type of error which was encountered. It usually is one of the following:

BADCOUNT (=0)	a call has been made to a procedure with an invalid number of parameters specified.
BADVALUE (=1)	a procedure call has been made with an invalid parameter value.
HARDERR (=2)	a hardware I/O error has occurred (this is very rare).
OVERFLOW (=3)	a finite size array, such as a display list for *makeob(3G)*, has overflowed its boundary.
BADCALL (=4)	an illegal or unexpected procedure call.
BADARRAY (=5)	an array, such as a display list for *drawob(3G)*, does not have the correct internal format.
BADSTACK (=6)	an attempt has been made to push the PS2 hardware matrix stack more than 11 levels deep or to pop the stack when it is already at the top level.

The second number in the error message, *y*, denotes the procedure number in which the error occurred. The name may be found by scanning the procedure summary list below or, alternatively, invoking the *pserrs(1G)* shell script. This command reports both the mnemonic name for the type of error and the name of the procedure in which the error occurred. A stack trace may optionally also be produced.

An alternative error printout is also available. Calling the subroutine *xerrors(3G)* enables this expanded form of error reporting. This method is much preferred over the numerical method detailed above, but consumes additional main memory space for storage of messages. When the host is a VAX processor and memory space is not a critical resource, this method of error reporting is selected automatically.

F77 DIFFERENCES

The synopsis for usage of the individual procedures is typically oriented toward the C programmer. To call these same routines from an f77 program it usually suffices to distinguish between which are functions and which are true subroutines, inserting the word "call" in front of the latter. Also, a limitation of six characters per subroutine name is silently enforced by the f77 compiler so that routines known by longer names in C must be truncated to six characters in f77 programs (e.g. "autocur(1);" in C becomes "call autocu(1)" in f77). Lastly, many global variables accessable in C are inaccessable in f77. To get around this limitation either optional arguments must be supplied to the particular subroutine or a simple interface procedure must be written in C. For example, in f77 there is no way to access the global character array *_pskbrd* in connection with the *getchr* subroutine, although by supplying the optional *bufp* and *size* arguments the desired data is made equivalently available. This last limitation could be alleviated at the expense of restricted common block names, although this is currently considered undesirable.

FILES

/usr/include/ps.h	C "include" file
/usr/lib/libg.a	C ps2gsp library
/usr/lib/libgf.a	f77 ps2gsp library
/usr/lib/libgu.a	C utility library
/usr/lib/libguf.a	f77 utility library

SEE ALSO

psinit(3G), pserrs(1G), intro(3), ld(1), cc(1), f77(1),
Picture System 2 User's Manual
Principals of Interactive Graphics, W.M. Newman and R.F. Sproull, 2nd edition

ACKNOWLEDGEMENTS

Mickey Mantle of Evans and Sutherland and Authur Olson of the University of California, San Diego provided much insite into the operation of the Picture System and were instrumental in the original release of this software package; Mickey's patience during countless number of telephone calls is especially appreciated. Ollie Jones of UCSF provided many of the general purpose routines for the Graphics Utility Library. Conrad Huang, also of UCSF, did most of the conversion for the f77 version of the package. Funding for this project was provided to Robert Langridge, principle investigator, by the National Institutes of Health, Grant #RR-1081.

Thomas E. Ferrin, February 1977

SUBROUTINE SUMMARY

The following is a list of subroutines currently available:

Picture System 2 Graphics Subroutine Package Summary			
Error #	Calling Sequence	Error #	Calling Sequence
42	analog(knob)	7	master(l, r, b, t [,w]) --- 2D
51	apndps( )	7	master(l, r, b, t, h, y [,w]) --- 3D
29	bldcon(type [,&matrix])	11	move(x, y [,z])
206	bldps(type, ps2loc)	11	moveto(x, y [,z])
19	blink(status)	27	nufram( )
17	charsz(size, tilt)	10	pop( )
30	cossin(angle, &cosine, &sine)	28	psbuf(nbuffers)
24	cursor([&x, &y [,&ipen]])	-	psfini( )
20	dash(status)	1	psinit([refresh, update])
13	dot(x, y [,z])	9	push( )
13	dotat(x, y [,z])	36	rbtmem(&array, maxlen, &lenptr [,&fulsub])
201	dowrbuf(&buffer, size)	36	rdtc(ps2loc, &pdploc)
14	draw2d(&data, npoints, fsm1, fsm2, z [,w])	4	rot(angle, axis)
15	draw3d(&data, npoints, fsm1, fsm2 [,w])	-	rsetps( )
16	draw4d(&data, npoints, fsm1, fsm2)	6	scale(x, y, z [,w])
35	drawob(&array)	22	scopes(number)
54	drawps(name)	37	setlit(number, status)
37	fswitch([group])	55	setps(memlimit [,nobjects, &array])
43	getchr([&buffer, &count])	57	setrot(ps2loc, angle, axis)
30	getrot(&array, angle, axis)	58	setscl(ps2loc, scalex, scaley, scalez [,w])
32	getscl(&array, scalex, scaley, scalez [,w])	59	settrn(ps2loc, tranx, trany, tranz [,w])
31	gettrn(&array, tranx, trany, tranz [,w])	208	sia(flag)
26	hitest(status)	21	speed(value)
25	hitwin(x, y, size [,w])	34	stopob( )
21	huesat(hue [,sat])	52	stopps( )
8	inst(l, r, b, t [,h [,y [,w]]])	36	stopwb( )
205	ispchd([&ipen])	53	subps(name)
37	iswset(number)	23	tablet([&x, &y [,&ipen]])
56	jumpps(name)	18	text([nchars,] &string)
37	lights(value [,group])	5	tran(x, y, z [,w])
12	line(x, y [,z])	20	txture(linetype [,cont])
12	lineto(x, y [,z])	2	vwport(l, r, b, t, h, y)
33	makeob(&array, maxlen, &lenptr [,&fulsub])	36	wbtmem(type)
51	makeps(name, &lenptr [,&fulsub])	3	window(l, r, b, t [,w]) --- 3D orthogonal
51	maksps(name, &lenptr [,&fulsub])	3	window(l, r, b, t, h, y [,e [,w]])

NAME

analog – read the value of an analog channel

SYNOPSIS

analog(channel)
int channel;

DESCRIPTION

The *analog* function is called to read the current value of the specified analog channel and return the relative amount that the channel has changed since the last time *analog* was called to read that channel. This allows the values returned for a given channel to be accumulated in a variable and used for absolute positioning.

Channel is an integer which specifies the device channel number that is to be read. This value may be 0–7 for the eight Control Dials or 8–13 for the dual Joystick controls.

The value returned from *analog* is in the range of approximately ±32700 and is the relative amount that the channel has changed since it was last polled. *Analog* will return with a value of zero the first time it is called.

SEE ALSO

getknob(3GU)

NAME

angle, dihed – calculate angles

SYNOPSIS FOR C USAGE

double angle(a, b, c);
double a[3], b[3], c[3];

double dihed(a, b, c, d);
double a[3], b[3], c[3], d[3];

SYNOPSIS FOR FORTRAN USAGE

real angle(a, b, c)
real a(3), b(3), c(3)

real dihed(a, b, c, d);
real a(3), b(3), c(3), d(3)

DESCRIPTION

Angle returns the angle (in radians) between the vector *a*-*b* and the vector *b*-*c*.

Dihed calculates the "dihedral" angle between the planes defined by the points *a*, *b*, *c* and the points *b*, *c*, *d*.

SEE ALSO

acos(3M)

DIAGNOSTICS

If any two of the three points are the same (in the case of *angle)* or if any pair of the four points the same (in the case of *dihed)* , a divide check will occur.

NAME

apndps – re-initiate *makeps/maksps* mode of operation

SYNOPSIS

apndps()

DESCRIPTION

This routine is called to re-initiate the *makeps/maksps* mode of operation, thus allowing a Picture System 2 display list to be appended to dynamically. This always appends to the last object created by *makeps/maksps*.

SEE ALSO

makeps(3G)

NAME

bldcon – perform transformation operations and matrix manipulations

SYNOPSIS

bldcon(type [,matrix])
int type;
ps_t matrix[4][4];

DESCRIPTION

Subroutine *bldcon* is called to perform all Picture System 2 transformation operations and matrix manipulations.

Matrix is a type 'ps_t' array (16 elements in length) which is used as specified by argument 1. This argument is used only for those operations which need an input matrix (operations 1, 2, 3 and 6).

Type is an integer which specifies the type of call. Valid values for *type* and the operation performed for each are:

INITIALIZE (=0)	Initialize Matrix Stack pointer and reset the stack length.
LOADMAT (=1)	Load the Transformation Matrix from the 16-element array specified by argument 2.
CONCATMAT (=2)	The 16-element array specified by argument 2 is post-multiplied by the existing Transformation Matrix. The resulting compound matrix is stored as the new Transformation Matrix.
STOREMAT (=3)	Store the Transformation Matrix into the 16-element array specified by argument 2.
POPMAT (=4)	Pop the top element of the Matrix Stack into the Transformation Matrix.
PUSHMAT (=5)	Push the Transformation Matrix onto the Matrix Stack.
PRECONMAT (=6)	The current Transformation Matrix is pushed onto the Matrix Stack and the *transpose* of this matrix is pre-multiplied by the 16-element array specified by argument 2. The resulting matrix is again transposed and stored as the new Transformation Matrix; this effectively *pre*-multiplies a transposed input matrix by the current Transformation Matrix. The original Transformation Matrix is left stored on the stack.

NOTE

The actual data sent to the Matrix Arithmetic Processor consists of an 'RSR' command word followed, if necessary, by the 16-element matrix data. For the LOADMAT command an additional data word is output immediately after the 'RSR' command word and before the matrix data. It is important to bear this in mind when manipulating the *lenp* parameter for the *makeob(3G)* and *makeps(3G)* subroutines.

SEE ALSO

push(3G), pop(3G), transpose(3GU), 'MATCON' command in PS2 Reference Manual

NAME

bldps – perform transformation matrix manipulations

SYNOPSIS

bldps(type, ps2loc)
int type;
psaddr_t ps2loc;

DESCRIPTION

The *bldps* subroutine is called to perform transformation and matrix manipulations in much the same way as the *bldcon* subroutine. For *bldps*, however, the 4x4 matrix data is located in Picture System 2 memory rather than PDP-11 memory. *Type* is an integer which specifies the type of call. Valid values for *type* and the operation performed for each are:

LOADMAT (=1)	Load the Transformation Matrix.
CONCATMAT (=2)	Concatenate the Transformation Matrix.
STOREMAT (=3)	Store the Transformation Matrix.

Ps2loc is the location in Picture System 2 memory of the 4x4 matrix to be used as specified by the *type* parameter.

SEE ALSO

bldcon(3G)

NAME

blink – make all subsequent lines blink

SYNOPSIS

blink(status)
int status;

DESCRIPTION

Subroutine *blink* is called to set the Line Generator status such that all subsequent lines drawn will blink or will not blink, dependent upon the value of the supplied parameter.

If *status* is 0, then blink mode is turned off, otherwise it is turned on.

SEE ALSO

txture(3G)

NAME

chargn – define a PS2 character set

SYNOPSIS

chargn [fontfile]

DESCRIPTION

Chargn is an interactive graphics program that allows the user to define a character set for the Picture System 2 programmable character generator. Details of the character generator hardware may be found in the Picture System 2 Reference Manual, Section 2.4.4.

When *chargn* is invoked it initially displays two menus. The upper 'character' menu is used to display the current individual character definitions and select the desired character(s) for modification. *Fontfile*, if given, is the name of a file containing the character definitions of a existing character set. The lower 'function' menu selects which editing function is to be performed. The functions are:

exit Terminate the program.

init1 Delete all characters in the current character set and re-initialize *chargn* for a single-font character set.

init2 Delete all characters in the current character set and re-initialize *chargn* for a two-font character set.

font1 Select the first font of a character set for editing.

font2 Select the second font of a two-font character set for editing. If *chargn* was initialized for a single character font, the message *'font 2 is non-existent'* is displayed.

read Read the given character set from disk and merge it with the current character set. Any character in the current set that has a cross displayed over it is discarded and replaced with the corresponding character from the new character set. If the number of fonts in the new character set does not match the number of fonts originally specified the message *'incorrect number of fonts'* is displayed.

write Write the current character set to disk. The filename is specified via the Picture System keyboard.

copy Copy one character definition to another character's position. A master character must be selected followed by an instance character. This mode continues until a new function is selected.

swap Interchange the two selected characters in the character set. This mode continues until a new function is selected.

discard Specify which characters will be discarded when a new character set is read from disk and merged with the current character set. Any characters already marked for discard (i.e. have a cross drawn over them) are restored before the newly selected characters are marked for discard. Prompting for characters continues until another function is selected.

delete Specify additional characters for discard. Currently discarded characters are not restored.

edit Create a new character or edit an existing character. A particular character must be selected and a new set of menus is displayed on the screen. A 16 x 16 *character generation definition grid* is used to define the relative position of the next character stroke. The menu items are used to define the type of command for the next stroke. The commands are:

chdraw Use the *chdraw* instruction in the definition of a character. The user must select a point on the definition grid to specify the offset for the draw. A line will be drawn from the origin of the grid to the point selected and the grid will then move so that its origin is located at the selected point. The user may then specify the offset for the next draw. *Chargn* will remain in this mode until another command is selected.

chmov Use the *chmov* instruction in the definition of a character. Operation is analogous to the *chdraw* command.

rmov Use the *rmov* instruction in the definition of a character. When this command is selected, the grid will move so that its origin is located at the point in the character last specified by the previous *rmov* instruction or, if one doesn't exist, the beginning of the character. The user must select a point on the definition grid to specify the offset for the move and the grid will then be located so that its origin is at the selected point. *Chargn* will remain in this mode until another command is selected.

rmove Use the *rmov* instruction with the *end* bit set in the definition of a character. All characters generated with *chargn* must end with an *rmove* instruction. Operation is analogous to the *rmov* command except *chargn* will exit *rmove* mode after the desired endpoint is selected.

exit Place the current character definition into the character set and terminate *edit* mode. If the *exit* command is selected and the character does not end with an *rmove* instruction, the message *'character must end with an rmove'* is displayed and *edit* mode is not terminated.

delete Delete the last instruction from the character definition. The grid will move so that its origin is at the end position of the previous instruction. If the previous instruction was a *chdraw*, the line specified by that instruction will be removed from the screen. *Chargn* returns to the previously selected mode after the character instruction is deleted.

The files */usr/src/ps2gsp/ps7/c128* and */usr/src/ps2gsp/ps7/c256* contain the definitions of the standard control characters (characters 0 through 37 octal) for character sets of one and two fonts, respectively. These files may be read using the *read* command or by specifying them as *fontfile*.

SEE ALSO

cload(3G), Picture System 2 Reference Manual, Section 2.4.4

NOTE

Control characters should be modified with care. If a control character is selected, the warning message *'Warning! control character'* is displayed and the same character must be immediately selected a second time before modification is permitted.

DIAGNOSTICS

If modification to a character causes the length of the character set to exceed 1024 words, the modification will not be accepted and the message *'character set exceeds 1024 instructions'* will be displayed.

BUGS

Characters 0, 3 and 4 cannot be modified. Existing character sets optimized with *jms/jmp* instructions may not be modifiable.

NAME

charsz – update character size, font and orientation

SYNOPSIS

charsz(size, font)
int size;
struct {char orient; char type; } font;

DESCRIPTION

The *charsz* subroutine is called to update the character size, orientation and font selection parameters used by the Character Generator.

Size is an integer which specifies the character size to be selected. Valid values for *size* are:

0 = 0.36 cm (0.14 inches)
1 = 0.08 cm (0.03 inches)
2 = 0.15 cm (0.06 inches)
3 = 0.25 cm (0.10 inches)
4 = 0.40 cm (0.16 inches)
5 = 0.68 cm (0.27 inches)
6 = 1.14 cm (0.45 inches)
7 = 1.88 cm (0.74 inches)

These sizes give the approximate height of a capital letter (A-Z,0-9) based upon a 28.6 x 28.6 cm (11.2 x 11.2 inch) screen viewing area. It should be noted that subscript and superscript characters are only available for *size* = 2 to 7. Subscript or superscript character codes (30-33 octal) used when *size* = 0 or 1 will result in an invalid character size selection. *Font.orient* is a variable which specifies the desired character orientation. Valid values for *font.orient* are:

0 = Horizontal character orientation.
1 = 90 degree counterclockwise character orientation.
2 = Italic 90 degree counterclockwise character orientation.

Font.type is a variable which specifies one of four available character fonts, two of which reside in Character Generator read-only-memory (ROM) and two of which reside in the user programmable area of Character Generator memory (RAM). The valid values for *font.type* are:

0 = The normal character set.
1 = The 'fast font' character set.
2 = The low order RAM character set.
3 = The high order RAM character set.

(See the various 'defines' available in the file *ps.h* for mnemonics of the codes listed above. Note that after powering up the Picture System 2 the RAM memory will contain garbage and must be loaded with a user specified character set.)

SEE ALSO

text(3G), getchr(3G), psinit(3G), cload(3G)

NAME

cload – character generator support

SYNOPSIS

```
cload(font)
char font[2048];

cfpush(hi, lo)
int hi, lo;

cfpop()

cginit()
```

DESCRIPTION

These routines are used to program the Picture System 2 Character Generator:

Cload takes the given character font and loads it into character generator RAM memory.

Cfpush pushes and then loads the 16-level font parameter stack; *hi* is the high-order six bits of the font parameter data (intensity and character style) and *lo* is the low-order six bits (coefficient address). The coefficient offset is set to zero.

Cfpop restores the previous entry in the font parameter stack.

Cginit reinitializes the character generator to its power-up state.

SEE ALSO

chargn(1G), Picture System 2 reference manual, section 2.4.4 (pgs 2–178 to 2–204)

NAME

cossin – compute cosine and sine for an angle

SYNOPSIS

cossin(angle, cos, sin)
int angle;
ps_t *cos, *sin;

DESCRIPTION

The *cossin* function is called to compute a cosine and sine for the angle specified and returns these values to the calling routine as a binary fraction (the form expected by the Picture Processor). *Cossin* is useful for forming one's own rotation matrices for use in updating a Linear Display List.

Angle is an integer which specifies that angle of rotation. The angle is given by dividing a circle into 2**16 equal parts, with zero being equal to zero degrees and 2**15–1 equaling 180 degrees. Two's complement addition, ignoring overflow, causes the angle to increase counter-clockwise through 360 degrees, when viewed along the specified axis in the positive direction. *Cos* and *sin* are the addresses of 'ps_t' type variables in which the computed cosine and sine, respectively, will be returned.

SEE ALSO

rot(3G), getrot(3G)

NAME

cursor – display a cursor

SYNOPSIS

cursor([x, y [,pen]])
int *x, *y, *pen;
and
autocur(status)
int status;

DESCRIPTION

The *cursor* subroutine is called to display a cursor at the position specified by the parameter list. As an alternative, the user may specify initiation of automatic cursor mode via *autocur*. This will cause a cursor to be displayed upon each frame refresh regardless of the new frame update rate. The cursor displayed in automatic cursor mode will be at the position determined by the data tablet cursor and is displayed within a full screen viewport. In either case the cursor displayed consists of a cross whose center is at the desired X and Y position.

In the call to *cursor*, *x* and *y* are the addresses of integers which specify the X and Y cursor positions. If these arguments are omitted the default external variables *_ix*, *_iy* and *_ipen* are used. The values of *x* and *y* should be in the approximate range of ±32767. *Pen* is the address of integer which, if specified, should be the pen information which is returned from the *tablet* subroutine. The specification of this parameter allows the cursor to be increased in intensity whenever the pen is down, providing visual feedback of the pen status.

When calling the *autocur* subroutine, if *status* $\neq 0$, automatic cursor mode is turned on, otherwise automatic cursor mode is turned off.

SEE ALSO

tablet(3G)

NAME

dist3 – distance between two 3-dimensional points

SYNOPSIS FOR C USAGE

double dist3(a, b)
double a[3], b[3];

SYNOPSIS FOR FORTRAN USAGE

real dist3(a, b)
real a(3), b(3)

DESCRIPTION

Dist3 calculates the distance between two three-dimensional points (the second-degree norm of their vector difference).

NAME

dot – draw a dot at a relative coordinate

SYNOPSIS

dot(dx, dy [,dz])
int dx, dy, dz;

DESCRIPTION

The *dot* subroutine is called to draw a dot at the specified 2D relative X, Y coordinates or the 3D relative X, Y and Z coordinates from the current position.

Dx, dy and *dz* are integers which specify the delta X, Y and Z relative coordinates. If *dz* is not specified, the 3-space relative coordinate *(dx, dy, 0)* is used for positioning instead.

SEE ALSO

dotat(3G)

NAME

dotat – draw a dot at an absolute coordinate

SYNOPSIS

dotat(x, y [,z])
int x, y, z;

DESCRIPTION

The *dotat* subroutine is called to draw a dot at the 2D absolute X, Y coordinates or the 3D absolute X, Y, Z coordinates specified.

X, *y* and *z* are integers which specify the X, Y and Z absolute coordinate. If *z* is not specified, the 3-space point (x, y, 0) is used for positioning instead.

NOTE

Dotat positions the dot with the homogeneous coordinate (IW) = 32767.

SEE ALSO

dot(3G)

NAME

dowrbuf, nowrbuf – buffer text output

SYNOPSIS

dowrbuf(array, size)
ps_t *array;
int size;
and
nowrbuf()

DESCRIPTION

Write buffering is a technique used to decrease the overhead associated with UNIX system calls. It consists of declaring an array to hold the data for several system calls and then passing the array of data with one call. With the potential for a large number of small word count data transfers to the Picture System, this technique saves a substantial amount of system overhead. Even on the PDP-11/70 each UNIX system call takes a minimum of 320 microseconds.

The code to implement write buffering has been incorporated into the Picture System software in such a way as to be transparent to the user. That is, once the user has declared the location and size of his/her write buffer he/she need not be concerned with it. The software will buffer output to the Picture System, flushing the buffer whenever necessary. This includes whenever the buffer becomes full, a *nufram* call is made, a *drawob* call is made with the display list larger than the write buffer array (the display list is transferred directly in this case), a store matrix command is executed, or an explicit call is made to *_flwrbuf* or *nowrbuf* (the latter disables the write buffering feature).

Array is a pointer to an array of type 'ps_t' in user memory space allocated for storage of the buffered data. *Size* is the number of ps_t array elements in the allocated space. For convenience, the BUFRPS(size) macro defined in <ps.h> may be used to automatically allocate buffer space.

Experience has shown that the optimal buffer size is 100 to 200 elements and that it is advantageous to use write buffering in nearly every graphics program which uses the Picture System 2.

NAME

draw2d – draw two-dimensional coordinate data

SYNOPSIS

draw2d(data, num, f1, f2, z [,w])
ps_t *data;
int num, f1, f2, z, w;

DESCRIPTION

The *draw2d* subroutine is called to draw two-dimensional data coordinate points using the drawing mode specified in the parameter list. The points to be drawn are arranged in X, Y pairs and are displayed at an intensity which is dependent upon both the *z* parameter and the intensity values previously specified for the hither and yon clipping planes.

Data is a type 'ps_t' array (2 * *num* elements in length) which contains the X, Y coordinate points to be drawn. These data will be drawn in the drawing mode specified by the arguments *f1* and *f2* at the intensity specified by argument *z*. *Num* is an integer which specifies the number of coordinate pairs to be drawn. *F1* is an integer which specifies the type of draw function to be performed. Valid values for *f1* are:

DISNEW (=0) Disjoint lines from new position.
DISCUR (=1) Disjoint lines from current position.
CONNEW (=2) Connected lines from new position.
CONCUR (=3) Connected lines from current position.
DOTTED (=4) Dot at each point.

F2 is an integer which specifies the mode in which the coordinates are interpreted. Valid values for *f2* are:

FIRSTABS (=0) Absolute-relative-relative-relative-etc.
RELATIVE (=1) Relative always.
ABSOLUTE (=2) Absolute always.
SETBASE (=3) Set base-offset-offset-offset-etc.
OFFSET (=4) Offset always.

Z is an integer which specifies the Z position of the X, Y coordinate pairs drawn. This Z position is used to compute the intensity of the drawn data. A value of $z = 0$ will produce lines of maximum intensity when drawn using a two-dimensional window (the maximum intensity is specified using the *vwport(3G)* subroutine). *W* is an integer used to scale the coordinate data. If the scale factor is omitted or given as zero, it is treated as 32767.

SEE ALSO

draw3d(3G), draw4d(3G), line(3G), move(3G), lineto(3G), moveto(3G)

NAME

draw3d – draw three-dimensional coordinate data

SYNOPSIS

draw3d(data, num, f1, f2 [,w])
ps_t *data;
int num, f1, f2, w;

DESCRIPTION

The *draw3d* subroutine is called to draw three-dimensional data coordinate points using the drawing mode specified in the parameter list. The points to be drawn are arranged in X, Y and Z triplets and are displayed at an intensity which is dependent upon both the *z* coordinate data and the intensity values previously specified for the hither and yon clipping planes.

Data is a type 'ps_t' array (3 * *num* elements in length) which contains the X, Y and Z coordinates points to be drawn. These data will be drawn in the drawing mode specified by the arguments *f1* and *f2*. *Num* is an integer which specifies the number of coordinate triples to be drawn. *F1* is an integer which specifies the type of draw function to be performed. Valid values for *f1* are:

DISNEW (=0) Disjoint lines from new position.
DISCUR (=1) Disjoint lines from current position.
CONNEW (=2) Connected lines from new position.
CONCUR (=3) Connected lines from current position.
DOTTED (=4) Dot at each point.

F2 is an integer which specifies the mode in which the coordinates are interpreted. Valid values for *f2* are:

FIRSTABS (=0) Absolute-relative-relative-relative-etc.
RELATIVE (=1) Relative always.
ABSOLUTE (=2) Absolute always.
SETBASE (=3) Set base-offset-offset-offset-etc.
OFFSET (=4) Offset always.

W is an integer used to scale the coordinate data. If the scale factor is omitted or given as zero, it is treated as 32767.

SEE ALSO

draw2d(3G), draw4d(3G), line(3G), move(3G), lineto(3G), moveto(3G)

NAME

draw4d – draw homogeneous coordinate data

SYNOPSIS

draw4d(data, num, f1, f2)
ps_t *data;
int num, f1, f2;

DESCRIPTION

The *draw4d* subroutine is called to draw homogeneous coordinate data using the drawing mode specified in the parameter list. The points to be drawn are arranged as sets of X, Y, Z and W coordinates and are displayed at an intensity which is dependent upon both the scaled *z* coordinates and the intensity values previously specified for the hither and yon clipping planes.

Data is a type 'ps_t' array (4 * *num* elements in length) which contains the X, Y, Z and W coordinate data to be drawn. These data will be drawn in the drawing mode specified by the arguments *f1* and *f2*. *Num* is an integer which specifies the number of coordinate sets to be drawn. *F1* is an integer which specifies the type of draw function to be performed. Valid values for *f1* are:

DISNEW (=0) Disjoint lines from new position.
DISCUR (=1) Disjoint lines from current position.
CONNEW (=2) Connected lines from new position.
CONCUR (=3) Connected lines from current position.
DOTTED (=4) Dot at each point.

F2 is an integer which specifies the mode in which the coordinates are interpreted. Valid values for *f2* are:

FIRSTABS (=0) Absolute-relative-relative-relative-etc.
RELATIVE (=1) Relative always.
ABSOLUTE (=2) Absolute always.
SETBASE (=3) Set base-offset-offset-offset-etc.
OFFSET (=4) Offset always.

SEE ALSO

draw2d(3G), draw3d(3G)

NAME

drawob – output a Linear Display List

SYNOPSIS

drawob(array)
ps_t *array;

DESCRIPTION

The *drawob* subroutine is called to output a Linear Display List, previously prepared by the *makeob* subroutine, to the Picture Processor. When this routine is called, the Picture System is placed in a mode such that the entire command/data list is processed in a single DMA operation.

Array is a user-supplied array in which the Linear Display List previously accumulated by *makeob* is located.

SEE ALSO

makeob(3G)

NAME

drawps – draw an object stored in Picture System 2 memory

SYNOPSIS

drawps(name)
int name;

DESCRIPTION

This routine is called to process a structured display list previously stored in Picture Memory. The Picture Processor is set to active input mode and starts reading data from the beginning address in Picture Memory where the specified object resides.

Name is the object identifier.

SEE ALSO

makeps(3G)

NAME

errcheck – correct for roundoff errors in rotation matrices

SYNOPSIS

errcheck(mat);
ps_t mat[4][4];

DESCRIPTION

Errcheck adjusts the given Picture System matrix so that the first three rows and columns (the portion of the matrix which applies to rotation of the image) are unitary. The rows and columns are alternately normalized.

A matrix which is the result of repeated Matrix Arithmetic Processor operations will accumulate roundoff error. If this subroutine is called at regular intervals (such as every tenth time the matrix is changed using the Matrix Arithmetic Processor), the roundoff error will be prevented from having an effect on the shape or scale of a picture displayed using that matrix.

SEE ALSO

dist3(3GU)

DIAGNOSTICS

If the matrix is not a Picture System rotation matrix, the results will be unpredictable. If all the elements of any row or column are zero, a divide check will occur.

NAME

fswitch – read Function Switches

SYNOPSIS

unsigned fswitch([group])
int group;

DESCRIPTION

The *fswitch* function returns the 16-bit value read from a particular group of Picture System 2 Function Switches. *Group* is an integer which specifies which group of Function Switches is to be read. If the group number is omitted, Function Switch set 1 is assumed. Valid values for *group* are:

1 = Function Switch & Light Group 1.
2 = Function Switch & Light Group 2.
3 = Function Switch & Light Group 3.
4 = Function Switch & Light Group 4.

SEE ALSO

iswset(3G), lights(3G), setlit(3G)

NAME

fulsub – subroutine to output segmented Picture System objects

SYNOPSIS

```
extern char *_obname;
#define OBLNGTH 256
ps_t ob[OBLNGTH];
int obl;
int fulsub();

_obname = "myfile";
 ...
makeob(ob, OBLNGTH, &obl, fulsub);
 ...
```

DESCRIPTION

Fulsub is a useful argument for the *makeob(3G)* and *rbtmem(3G)* Picture System routines. When used with *makeob*, it permits the object buffer in memory to be small and writes a disk file which can be read as a single array and passed directly to *drawob(3G)*, just as if it were originally assembled into a single large object array. When used with *rbtmem*, the disk file can be used as an argument to *pscopy(1G)*.

The global character pointer "_obname" must be initialized with the name of the output file somewhere in the source program. If this is not done the name "copy.tmp" will be used by default. If the file already exists the new data will be appended to the current file. This is meaningful to *pscopy* (since each data set will be treated as a separate plot), but *drawob* does not understand about multiple objects.

FILES

copy.tmp

SEE ALSO

pscopy(1G)

DIAGNOSTICS

Reports if the file cannot be created.

NAME

getchr – read from Picture System 2 keyboard

SYNOPSIS

```
getchr([bufp, size])
char *bufp;
int size;
extern char _pskbrd[];
extern char _pskbchr;
extern int _pskblen;
extern int _ignornull;
```

DESCRIPTION

The *getchr* subroutine is called to return characters typed in on the Picture System 2 keyboard. Although it is similar in spirit to the E&S subroutine of the same name, it is NOT compatible. The optional arguments to the UNIX version are *bufp* and *size* which are the address and length, respectively, of a user supplied buffer.

Normally *getchr* returns a zero value indicating no new input has occurred since the previous call. *Getchr* returns a negative value when it receives a new character (other than a terminating character) or when the currently buffered command line has changed due to erase or kill processing (see below). If a terminator character [carriage return (=015), line feed (=012), form feed (=014), or escape (=033)] is entered, *getchr* will return the number of characters entered since the previous terminator character (again dependent on erase and kill processing).

Currently, only one character is processed on each call to *getchr* so that this routine must usually be called several times to form a complete command string. The array *_pskbrd* is available and contains both the characters which have been entered since the last terminator and a blinking underscore character at the next available character position. This is useful as an argument to a *text* call and will display the currently entered keyboard characters and blinking cursor.

The following technique can be used to efficiently read characters from the keyboard:

> Within the display update loop of the program make a call to *getchr;* if the routine returns a zero value nothing involving this portion of the display has changed since the previous call to *getchr*. If *getchr* returns a non-zero value then the buffered command line has changed in some way (either by typing a valid character or through internal erase/kill processing) and at a minimum a *text(_pskbrd)* and a *nufram* call must be made to accurately reflect the new contents of the character buffer. In addition, if the value returned was positive then some user processing of the command line must usually be performed.
>
> A global (type 'int') variable *_ignornull* is also available and defaults to a non-zero value. This causes *getchr* to ignore two successive terminating characters. This will prevent false command string processing by the user program as the result of a hardware glitch which occurs somewhat frequently. Setting *_ignornull* to zero will cause *getchr* to return a non-zero value even if there were no other characters entered between two successive terminator characters.
>
> In any event, the last character entered from the keyboard (including the terminator character) is available for inspection in the variable *_pskbchr*, as is the current number of characters in the buffer in the variable *_pskblen*. If the *bufp* and *size* arguments are supplied, this buffer is used instead of *_pskbrd*.

Lastly, *getchr* does character-erase (=ctrl h or rubout), word-erase (=ctrl w), and kill (=ctrl u) processing on the entered characters similar to standard UNIX processing.

SEE ALSO

charsz(3G), text(3G)

NAME

getknob – get values of interactive knobs

SYNOPSIS

getknob(chan, mode);
int chan, mode;

DESCRIPTION

Getknob provides access to the interactive knobs and joysticks of the Picture System. It returns the value associated with the knob or joystick numbered *chan*, calculated by the method specified with *mode*.

A *mode* of zero causes the values of all channels from all devices to be read into core. This must be done initially and at regular intervals in order to obtain updated values from the devices.
A *mode* of 1 causes *getknob* to return the absolute knob position of the specified channel.
A *mode* of 2 returns the relative amount the knob setting has changed since the previous call.
A *mode* of 3 zeroes the absolute positions of all the knobs (i.e. sets the current position as the new "home" position).

The possible values of *chan* and the associated devices are:

Channel	Device
0-7	Interactive Knobs
8	Joystick #1, X
9	Joystick #1, Y
10	Joystick #1, Z
11-13	Joystick #2
14-15	Unused

SEE ALSO

analog(3G)

FILES

/dev/ps.cdj

DIAGNOSTICS

same as those reported by *analog*.

NAME

getrot, setrot – build a rotation transformation

SYNOPSIS

getrot(array, angle, axis)
ps_t array[4][4];
int angle, axis;
and
setrot(psmemloc, angle, axis)
psaddr_t psmemloc[4][4];
int angle, axis;

DESCRIPTION

The *getrot* and *setrot* subroutines are called to build a rotation transformation based on the angle and axis of rotation specified in the parameter list. The transformation is then returned in the user-supplied 16-element matrix buffer, *array*, or in the case of the *setrot*, stored in Picture System 2 memory.

Array is a 16-element array of type 'ps_t' in which the 4x4 rotation transformation is to be returned. *Psmemloc* is the 16-element array location in Picture Memory. *Angle* is an integer which specifies the angle of rotation. The angle is given by dividing a circle into 2**16 equal parts, with zero being equal to zero degrees and 2**15–1 equaling 180 degrees. Two's complement addition, ignoring overflow causes the angle to increase counter-clockwise through 360 degrees, when viewed along the specified axis in the positive direction. *Axis* is an integer which specifies the axis of rotation. Valid values for *axis* are:

XAXIS (=1) rotation about x axis.
YAXIS (=2) rotation about y axis.
ZAXIS (=3) rotation about z axis.

NOTE

The Picture System 2 software is designed for a left-handed coordinate system.

SEE ALSO

gettrn(3G), getscl(3G), makeob(3G), makeps(3G), rot(3G)

NAME

getscl, setscl – build a scaling transformation

SYNOPSIS

getscl(array, sx, sy, sz [,w])
ps_t array[4][4];
int sx, sy, sz, w;
and
setscl(psmemloc, sx, sy, sz [,w])
psaddr_t psmemloc[4][4];
int sx, sy, sz, w;

DESCRIPTION

The *getscl* and *setscl* subroutines are called to build a scaling transformation based on the X, Y and Z scaling terms specified in the parameter list. The transformation is then returned in the user-supplied 16-element matrix buffer, *array*, or, in the case of *setscl*, stored in Picture Memory.

Array is a 16-element array of type 'ps_t' where the 4x4 scaling transformation is to be returned. *Psmemloc* is the 16-element array location in Picture System 2 memory. *Sx, sy* and *sz* are integers which specify the X, Y and Z scale values. *W* is an integer which specifies the factor used to scale the scaling definition values. If the scale factor is omitted or given as zero, it is treated as 32767.

SEE ALSO

getrot(3G), gettrn(3G), makeob(3G), makeps(3G), scale(3G)

NAME

gettrn, settrn – build a translation transformation

SYNOPSIS

gettrn(array, tx, ty, tz [,w])
ps_t array[4][4];
int tx, ty, tz, w;
and
settrn(psmemloc, tx, ty, tz [,w])
psaddr_t psmemloc[4][4];
int tx, ty, tz, w;

DESCRIPTION

The *gettrn* and *settrn* subroutines are called to build a translation transformation based on the X, Y and Z translational values specified in the parameter list. The transformation is then returned in the user-supplied 16-element matrix buffer *array*, or, in the case of *settrn*, stored in Picture System 2 memory.

Array is a 16-element array of type 'ps_t' where the 4x4 translation transformation is to be returned. *Psmemloc* is the 16-element array location in Picture Memory. *Tx, ty* and *tz* are integers which specify the scaled X, Y and Z translation values. *W* is an integer which specifies the factor used to scale the translational values. If the scale factor is omitted or given as zero, it is treated as 32767.

SEE ALSO

getrot(3G), getscl(3G), makeob(3G), makeps(3G), tran(3G)

NAME

hittag, hitset, hitclr – display list hit testing

SYNOPSIS

hittag(id)
int id;

hitset(x, y, size [,w])
int x, y, size, w;

hitclr()

DESCRIPTION

This set of routines implements a strategy for performing "hit" testing on objects stored as untransformed display lists (either in Picture System memory or PDP-11 memory). Identification "tags" are used to denote various sections within a display list. Each time *hittag()* is called a new identifier is stored in the current display list. After display list generation is complete, *hitset()* is called to specify a hit window and enable hit processing. *Drawps(3G)* or *drawob(3G)* then processes the specified display list(s) and the Matrix Arithmetic Processor determines if any data within this display list passes within the given hit window. *Hitclr()* is called to disable hit testing and determine if any hits have occured; if there has been a hit it returns a non-zero value corresponding to the tag identifying the relevent section of the display list.

When specifing the hit window, *x* and *y* are integers which specify the hit window X and Y coordinates. These values should be in the approximate range of ±32700. Normally these values are obtained from the X and Y position coordinates of the data tablet cursor. *Size* is an integer which specifies the hit window half size. This parameter is used to determine whether lines pass within a given distance *(size)* of the specified point *(x, y)*. *W* is an integer used to scale the hit window parameters. If the scale factor is omitted or given as zero, it is treated as 32767.

The *hittag/hitset/hitclr* mechanism allows complete display lists to be processed as single units in an efficient manner. In summary, the important steps are:

1. Generate the display list, tagging the relevent parts of the untransformed object with *hittag* identifiers.
2. Call *hitset* to set up the desired hit window and enable hit processing.
3. Call *drawps* or *drawob* to process the display list.
4. Call *hitclr* to disable hit processing. A non-zero return from *hitclr* corresponds to the tag of the "hit" object.

SEE ALSO

hitwin(3G), hitest(3G), makeps(3G), makeob(3G)

BUGS

Only the first identifier is returned for each hit in a display list; this is a limitation of the PS2 hardware.

NAME

hitest – hit testing

SYNOPSIS

hitest(status)
int status;

DESCRIPTION

The *hitest* function is called to determine if any data has passed within a prespecified hit window (see *hitwin(3G)*). The procedure for this test is of the form:

1. Call *hitwin* to set up the desired hit window.
2. Draw data *(draw2d, draw3d,* etc.) for comparison against that window.
3. Call *hitest* to determine if there was a 'hit'.
4. Repeat steps 2 and 3 as often as necessary, setting the *hitest* status argument to a nonzero value on the last call.

Hitest returns a zero value if there has been no hit, and a non-zero value if there has been a hit. *Status* is an integer supplied by the user which indicates whether the hit testing has been completed. *Status* = 0 indicates an intermediate hit test and *status* $\neq$ 0 indicates the final hit test for this set of data.

SEE ALSO

hitwin(3G)

NAME

hitwin – specify a hit window

SYNOPSIS

hitwin(x, y, size [,w])
int x, y, size, w;

DESCRIPTION

The *hitwin* procedure is called to specify a window through which data will be drawn and tested for a 'hit'. A window transformation of the specified size and coordinates is created and preconcatenated with the current transformation in the Picture Processor after first saving the original transformation matrix. The Picture Processor status is set to prohibit data from being stored into the refresh buffer while hit testing is in progress. A call to *hitest* with a non-zero argument restores the original transformation matrix and resets the Picture Processor status to its previous state.

X and *y* are integers which specify the hit window X and Y coordinates. These values should be in the approximate range of ±32700. Normally these values are obtained from the X and Y position coordinates of the data tablet cursor. *Size* is an integer which specifies the hit window half size. This parameter is used to determine whether lines pass within a given distance *(size)* of the specified point *(x, y)*. *W* is an integer used to scale the hit window parameters. If the scale factor is omitted or given as zero, it is treated as 32767.

SEE ALSO

hitest(3G), tablet(3G)

NAME

huesat – specify color and saturation

SYNOPSIS

huesat(hue [,sat])
int hue, sat;

DESCRIPTION

Huesat is called to specify the color for all subsequently drawn data. The parameters passed represent the hue (or color value) and optionally the saturation. When full saturation is selected (*sat*=7), the hue value alone selects the desired color. As the value of the *hue* parameter changes from 0 though 32, the color will range from green (*hue*=0) to cyan (*hue*=8) to blue (*hue*=16) to magenta (*hue*=24) to red (*hue*=32). As the value of the *hue* parameter changes from 32 though 63, the color will range from red (*hue*=32) to yellow (*hue*=48) to almost green (*hue*=63). As the saturation is reduced from the maximum value (*sat*=7 to *sat*=0) each color will become more pastel. For both *hue* = 0 and *sat* = 0 white is displayed.

Hue is an integer which specifies the color in which all subsequent data is displayed (0–63).

0 = Green
16 = Blue
32 = Red
48 = Yellow
63 = almost Green

Sat is an integer which specifies saturation (0–7). If this parameter is omitted full saturation is assumed.

0 = White
7 = Full saturation

NOTE

The Line Generator is automatically set to half speed when *huesat* is called. This is a requirement for the color monitor.

SEE ALSO

speed(3G)

NAME

inst – generate instancing transformations

SYNOPSIS

inst(nl, nr, nb, nt [,w])
int nl, nr, nb, nt, w;
and
inst(nl, nr, nb, nt, nh, ny [,w])
int nl, nr, nb, nt, nh, ny, w;

DESCRIPTION

The *inst* subroutine concatenates a two- or three-dimensional instancing transformation to the Picture Processor Transformation Matrix. This subroutine is used, in conjunction with the *master* subroutine, to produce multiple instances of an object or symbol. For each desired appearance of the object, the *inst* subroutine is called to specify the location (and implicitly the size) of that appearance; then the user-supplied routine describing the object is called to display the object previously defined within a two-dimensional or three-dimensional enclosure. The *inst* subroutine pushes the initial Transformation Matrix onto the Transformation Stack before concatenating the instancing transformation. In this way the original transform may be restored (POPped) by the user after the object has been drawn.

Nl, nr, nb, nt, nh and *ny* are integers which specify the scaled instance left, right, bottom, top, hither, and yon boundaries, respectively, in definition space coordinates (range = ±32767). For two-dimensional instancing, the window front, or hither, boundary is 0 and the rear, or yon, boundary is equal to *w*. *W* is an integer used to scale the instance boundaries. If this scale factor is omitted or given as zero it is treated as 32767.

SEE ALSO

master(3G), pop(3G)

NAME

ispchd – is pen changed?

SYNOPSIS

ispchd([penadr])
int *penadr;

DESCRIPTION

Ispchd is a function which may be used to determine whether the status of the data tablet cursor (pen) has changed in relation to the last time this routine was called. This facilitates the testing for pen transitions (i.e., up to down, down to up), a function often required in tablet interaction.

Penadr is the address of an integer variable which contains the pen information returned by the *tablet* subroutine. If omitted, the default external variable *_ipen* is used. Values returned by *ispchd* are:

PIUWU (=0) if pen is up and was up last call.
PIDWU (=1) if pen is down and was up last call.
PIDWD (=2) if pen is down and was down last call.
PIUWD (=3) if pen is up and was down last call.

SEE ALSO

tablet(3G)

NAME

iswset – is switch set?

SYNOPSIS

iswset(n)
int n;

DESCRIPTION

Iswset (Is SWitch SET) is a function which may be used to determine whether a particular switch of the Picture System 2 Function Switches is set. *Iswset* returns 1 if the switch is set and zero if it's not set.

N is an integer which specifies the switch number that is to be tested.

N = 0-15 for 1 set of Function Switches & Lights
N = 0-31 for 2 sets of Function Switches & Lights
N = 0-47 for 3 sets of Function Switches & Lights
N = 0-63 for 4 sets of Function Switches & Lights

SEE ALSO

fswitch(3G), setlit(3G), lights(3G)

NOTA BENE

Iswset is acceptable efficiency-wise if one or two switches are being tested, but fswitch(3G) is much more efficient if several switches must be tested within the same display loop.

NAME

joystick – simulated interactive joystick device

SYNOPSIS

joystick(x, y, size, xval, yval, xret, yret, flag [, label [, mkob]])
int x, y;
int size;
int xval, yval;
int *xret, *yret;
int flag;
char *label;
int mkob;

DESCRIPTION

Joystick provides a simulation of a two-dimensional joystick device. The simulation makes use of a defined area on the Picture System to represent two analog variables and uses the tablet for interaction with those variables. A ''star-shaped'' object is drawn to describe the location and center of the area.

The parameters used during the subroutine call specify a ''joystick area'' of size *size* and location *x, y* (upper left corner), and arrange for the return of a pair of differential values (pointed to by *xret* and *yret)* dependent upon the tablet cursor location within the joystick area and whether or not the cursor button is depressed. Optionally, (depending on the *flag* variable) a marker may be displayed at location *xval, yval* within the joystick area.

Size has the value 1 or 2; 1 indicates an area of 4K X 4K, and 2 an area of 8K X 8K (in absolute Picture System coordinates). The variables pointed to by *xret* and *yret* will be nonzero only if the cursor is within the limits of the joystick area and the cursor button is depressed. Their values represent the distances of the cursor from the center of the joystick area (in the X and Y directions).

Label points to a character string which is displayed just below the joystick area on the screen. If this argument is omitted no label is drawn.

The joystick and the marker (if displayed) become noticeably brighter when the cursor is inside the joystick's boundaries. This verifies a ''hit''.

The *mkob* parameter is used when the joystick is part of a PS2 display list. It should have a non-zero value when *joystick* is called during display list generation, and a zero value during normal display updates. The *label* parameter must be supplied if the *mkob* flag is used. If *mkob* is omitted, the entire joystick image is redrawn for each frame update.

The automatic cursor feature should be used in conjunction with this subroutine. This subroutine should be called once in each display loop. It does not call the *tablet* subroutine of the Picture System software directly, but assumes that it has been called in another part of the display loop.

SEE ALSO

pot(3GU), tablet(3G), cursor(3G)

BUGS

There is no way to distinguish the condition of the cursor being outside the joystick image area as opposed to located exactly over the center of the joystick.

The intensification of the joystick image when the cursor is over it depends on the setting of the contrast controls and on the Picture System viewport.

NAME

jumpps – jump to another Picture Memory object

SYNOPSIS

jumpps(name)
int name;

DESCRIPTION

This routine functions similar to *stopps(3G)*, but in addition causes the current object to 'jump' to the object specified, thus causing both objects to be output as a single unit.

Name is the object identifier.

SEE ALSO

makeps(3G), stopps(3G)

NAME

lights – set lights on Function Switches

SYNOPSIS

lights(value [,group])
int value, group;

DESCRIPTION

The *lights* subroutine is called to store a 16-bit value into a particular group of lights on the Picture System 2 Function Switches & Lights peripheral. *Value* is an integer which specifies the 16-bit value to be placed into the lights. *Group* is an integer which specifies which set of Picture System 2 Function Switches & Lights is to be used. If the group parameter is omitted Function Switch & Light group 1 is assumed. Valid values for *group* are:

1 for 1 set of Function Switches & Lights
1-2 for 2 sets of Function Switches & Lights
1-3 for 3 sets of Function Switches & Lights
1-4 for 4 sets of Function Switches & Lights

SEE ALSO

setlit(3G), fswitch(3G), iswset(3G)

NAME

line – draw a line in relative space

SYNOPSIS

line(x, y, [,z])
int x, y, z;

DESCRIPTION

The *line* subroutine is called to draw a line in the present line mode, specified during initialization or by a previous call to *txture* or *blink*, from the current position to the 2D relative X, Y coordinates or the 3D relative X, Y, Z coordinates specified. If *z* is not specified a line is drawn to the 3-space coordinate *(x, y, 0)*.

SEE ALSO

lineto(3G)

NAME

lineto – draw a line in absolute space

SYNOPSIS

lineto(x, y, [,z])
int x, y, z;

DESCRIPTION

The *lineto* subroutine is called to draw a line in the present line mode from the current position to the 2D absolute X, Y coordinates or the 3D absolute X, Y, Z coordinates specified. If *z* is not specified a line is drawn to the 3-space point *(x, y, 0)*.

NOTE

Lineto draws the line with the homogeneous coordinate (IW) = 32767.

SEE ALSO

line(3G)

NAME

lookat – produce lookat operators

SYNOPSIS

lookat(mat, matinv, a, b [,iw]);
int mat[4][4], matinv[4][4];
int a[3], b[3];
int iw;

DESCRIPTION

Lookat takes the three-dimensional vector *a-b* and produces a Picture System style transformation matrix, *mat*, which is equivalent to translating the point *a* to the origin and then rotating the vector *a-b* isometrically to the positive Z-axis. Rotation is performed first about the Y-axis to the Y-Z plane, and then about the X-axis to the X-Z plane. *Matinv*, the inverse of *mat*, is also produced.

Iw is an integer used to scale the coordinate data. If the scale factor is omitted it is treated as 32767.

If *matinv* is the inverse of *mat*, and *Z(n)* is an isometric rotation about the Z-axis by *n* degrees, it happens that the compound operator

mat • *Z(n)* • *matinv*

is equivalent to rotating *n* degrees about the vector *a-b*.

NAME

makeob – create a Linear Display List

SYNOPSIS

makeob(array, max, lenp [,fulsub])
ps_t *array;
int max;
int *lenp;
int fulsub();

DESCRIPTION

The *makeob* subroutine is called to initiate a mode in which all commands and data directed to the Picture Processor are intercepted and accumulated in a user-supplied main memory array in the form of a Linear Display List. The commands and data accumulated in this array may later be output to the Picture System as a single unit (or object), thus saving preparation time and other overhead. Most of the graphics subroutines described int this manual section (3G) may be used in creating a Linear Display List.

Array is a user-supplied array. *Max* is an integer which specifies the maximum number of elements in *array*. *Lenp* is the address of an integer variable where the number of array elements actually used will be maintained. This variable is set to 1 when *makeob* is called and again by each call to *fulsub* (see below). It may be (carefully) modified by the user if desired. *Fulsub* is a subroutine which, if specified, is called when *array* becomes full. If supplied, the *fulsub* calling sequence will be

*fulsub(array, *lenp, flag);*

The *flag* variable is zero for all but the final (terminating) call to *fulsub*.

The *lenp* parameter may serve as an extremely valuable tool where the *array* buffer is large enough to contain the entire object in one piece. In this case, if the value of *lenp* is saved immediately preceding the call to any Picture System 2 graphics subroutine, and the saved value incremented by one, it will serve as an *array* subscript pointing to the generated command word. If it is again incremented, it then points to the object data itself.

SEE ALSO

drawob(3G), getrot(3G), gettrn(3G), getscl(3G), makeps(3G), fulsub(3GU), bldcon(3G)

NAME

makeps, maksps – create a Picture Memory display list

SYNOPSIS

makeps(name,lenp [,fulsub])
int name;
int *lenp;
int fulsub();
and
maksps(name,lenp [,fulsub])
int name;
int *lenp;
int fulsub();

DESCRIPTION

This routine is called to initiate a mode in which commands directed to the Picture Processor are intercepted and stored in Picture System 2 memory, along with their associated data. The resulting 'object' may later be directed as a single unit to the PS2 matrix arithmetic processor via the *drawps(3G)* subroutine (for an object made by *makeps)* or invoked from another display list via the *subps(3G)* subroutine (for a sub-object made with *maksps).* This saves a significant amount of object preparation time and other system overhead. The Picture System must first have been initialized for *makeps/maksps* calls by means of the *setps(3G)* subroutine.

Name is an object identifier. Its legal range of values consists of any positive quantity up to 16 bits, including ASCII characters. If the object already exists it will be overwritten. There are restrictions on the length of the new object in this case (see below). *Lenp* is a pointer to an integer variable where the number of Picture Memory words actually used (by all such objects) will be maintained. When *makeps* or *maksps* is called, this variable is initially set to the location-1 in Picture Memory where the newly created object is to be stored.

The *lenp* parameter may serve as an extremely valuable tool. If the value of *lenp* is saved immediately preceding the call to any graphics subroutine, and the saved value incremented by one, it will serve as an index into Picture Memory for the 'RSR' command word. If it is again incremented, it then points to the command's object data. This value can be used to update the data via the *setrot, settrn* and *setscl* routines.

Fulsub is the address of a subroutine that is called if the space available for the current Picture Memory object becomes filled. This can occur for either of two reasons; if all objects stored in Picture Memory have exhausted the total available space allocated in the *setps* call, or, if an object is being re-written and has exhausted its local available space because another object is stored immediately after it. In the latter case, the current object cannot expand even though there may be memory available elsewhere (objects cannot be relocated).

Fulsub is called with two arguments. The arguments are the locations of the beginning and end of the current object in Picture Memory. The purpose of this subroutine call is to allow rational error recovery. The suggested course of action is to stop drawing and find more space.

As commands and data are collected in Picture Memory the maximum nesting of matrix pushes and pops is maintained in counters, and is included as part of the structure. In this way, it may be determined in advance if the display of an object will cause a matrix stack overflow or underflow, since software extension of the 8-deep matrix stack is not possible.

SEE ALSO

drawps(3G), getrot(3G), gettrn(3G), getscl(3G), makeob(3G), bldcon(3G)

NAME

master – generate master transformations

SYNOPSIS

master(ml, mr, mb, mt [,w])
int ml, mr, mb, mt, w;
and
master(ml, mr, mb, mt, mh, my [,w])
int ml, mr, mb, mt, mh, my, w;

DESCRIPTION

The *master* subroutine concatenates a two-dimensional or three-dimensional master transformation to the Picture Processor Transformation Matrix. This subroutine is used in conjunction with the *inst* subroutine for instancing of data. The master transformation is constructed from the values specified in the parameter list.

Ml, mr, mb, mt, mh and *my* are integers which specify the scaled master left, right, bottom, top, hither and yon boundaries in definition space coordinates (range = ±32767). For a two-dimensional master, the front, or hither, boundary is 0, and the rear, or yon, boundary is equal to *w*. *W* is the value used to scale the master boundaries. If the scale factor is omitted or given as zero, it is treated as 32767.

SEE ALSO

inst(3G)

NAME

menuset, menucheck, menu_box – Picture System menu package

SYNOPSIS

```
#include <ps.h>
int x, y;              /* upper left corner of the box */
unsigned xsize, ysize;     /* box width, box length */
int nx, ny;                /* number of windows in x and y directions */
char delim;                /* token delimiter within "str2" */
int item_pos;      /* index of item hit */
int mkob;                  /* non-zero during display list generation */
char *str2;                /* menu tokens, separated by "delim" */
char *str1;                /* text of hit menu item */
...
autocur();
for (;;) {             /* beginning of display loop */
        ...
        tablet();
        menuset(x, y, xsize, ysize, nx, ny [,delim]);
        ...
        item_pos = menucheck(str1, str2 [, mkob]);
        if (ispchd() == PIUWD) menu_box(item_pos);
        ...
        nufram();
}
...
```

DESCRIPTION

The Picture System menu package provides a method of displaying and using interactive ''light-buttons'' on the Picture System 2. This is implemented by means of items of text, called ''buttons'' or ''menu words'', displayed on the face of the CRT, and use of the data tablet and cursor to select these items. In order to use this package, the program must call the subroutines which set up the menu and which display and interrogate specific words each time through the display loop.

To reduce overhead and provide for a maximum of flexibility, calls to the tablet-control software (such as *tablet(3G)*) are left to the user, as is checking whether the cursor cross-hair button is depressed. Ordinarily (as in the above example) a particular event, such as releasing the button on the cursor, is used to trigger recognition of the of the particular menu item. (This recognition is *not* performed by the menu software.) Thus the program may define the menu interaction in any convenient way.

Menuset defines an area on the screen to be used as a menu area by future calls to *menucheck*. The top left corner of the area is given by the point *(x, y)* in absolute Picture System coordinates. The size of the area is defined as *xsize* * *ysize*. Note that these two parameters are of type ''unsigned'' to allow specification of an area larger than half the screen in either direction. This area is divided into a number of menu windows *(nx, ny)*, each of which will contain a single hittable item after *menucheck* is called. *Delim*, if supplied, defines the character which separates the menu words in the call(s) to *menucheck* to be something other than a space character.

The positions and sizes in the call to *menuset* will be used to directly drive the Picture System. Thus, before the call to *menucheck*, the screen ''window'' must be set so that the positions and sizes are meaningful, and the viewport must be set to full-screen. Also note that *menuset* must be called at least once before *menucheck*. It may be called several times in the same picture frame in order to define several menu areas on the screen.

Menucheck displays the menu words specified by *str2* in windows defined by the most recent call to *menuset*, and checks whether the tablet cursor is over any of the menu words displayed. If the tablet cursor is over any item, the index of that item is returned as the value of the *menucheck* function, and the text of

the word is returned in *str1*. The index of an item is defined as its position in the string pointed to by *str2;* the first item has a index of zero. If the tablet cursor is not over the any of the items, an index of -1 and a null string are returned.

The function makes no checks of character size, or string length. If more items are specified than windows available, the extra items will appear superimposed over the previous items. *Menucheck* may be called repeatedly once the initial *menuset* call is made; the new items will be drawn in the next available window positions.

The menu word will become noticeably brighter when the cursor is placed over it, providing visual feedback to the user.

The *mkob* parameter is used when the joystick is part of a PS2 display list. It should have a non-zero value when *menucheck* is called during display list generation; this will insure that all menu items are drawn at least once. During normal display updates *mkob* should have a zero value. Picture System draw commands will then only be output if a item has been hit. If this parameter is omitted, the entire menu image is redrawn for each frame update.

The returned value from *menucheck* may be used as an argument to *menu_box,* which will draw a box the size of an item window around the specified item. An argument of -1 to *menu_box()* is ignored.

SEE ALSO

tablet(3G), ispchd(3G)

BUGS

Depth cueing is assumed; also:

menuset:

Makes no check if any of the windows will go off the edge.
Has no idea of the size of characters.

menucheck:

No checks for menu word length are made.
The array pointed to by *str1* is assumed to be large enough to receive the menu word hit.
Extra delimiters (such as trailing blanks) will be taken as null items and will use an extra window.

menu_box:

Allows wrap around (this is a mixed blessing).

NAME

mmu – Memory Management Unit

SYNOPSIS

#include <ps2.h>

DESCRIPTION

This entry describes the UCSF Computer Graphics Laboratory homebrew memory management unit (MMU) for the Picture System 2.

The MMU extends the capacity of PS2 memory from 64K words to 256K words by adding an additional three banks of memory. The amount of memory directly addressable at any moment remains unchanged however, since the PS2 I/O bus is inherently only 16 bits wide.

The MMU logically consists of eight 2-bit registers. Each of the eight registers is associated with a particular PS2 device (e.g. MAP input controller) and the two bit value stored in a register determines which of the four possible memory banks is used in conjuction with memory accesses for this device. The top 256 addresses of all memory banks refer to the PS2 system control block and thus cannot be used for storage of object data.

When using the MMU, memory is most often partioned so that one bank of memory (bank 0) is used for refresh memory, and another bank of memory (bank 1) is used to store untransformed display lists. Thus the DIO port, DMA port and MAP input controller are set to access memory bank 1, while the MAP output controller and refresh controller are set to access memory bank 0. This allows very large displays to be generated without exhausting available memory for storage of the untransformed data.

In order to simplify implementation, the MMU physically consists of a single 16 bit register which is both readable and writable. Pairs of bits in this 16 bit word are associated with each PS2 device, beginning with the DMA I/O port in the low order bits. The register is cleared during a power up sequence and by a master reset of the PS2; this effectively disables the MMU.

Currently only banks 0 and 1 of extended memory are implemented. The possible PS2 devices are:

regno	mnemonic	PS2 device
0	X_DMAPORT	DMA port
1	X_RTI1	Remote terminal interface 1 (unused)
2	X_RTI2	Remote terminal interface 2 (unused)
3	-	Spare (unused)
4	X_DIOPORT	Direct I/O port
5	X_REFRESH	Refresh controller
6	X_MAPOUT	MAP output controller
7	X_MAPIN	MAP input controller

SEE ALSO

Extended Memory Multi-Picture System Maintenance Manual

NAME

move – move in relative space

SYNOPSIS

move(dx, dy [,dz])
int dx, dy, dz;

DESCRIPTION

The *move* subroutine is called to position to the specified 2D relative X, Y coordinates or the 3D relative X, Y, Z coordinates from the current position. *Dx, dy, dz* are the relative coordinates. If *dz* is not specified the 3-space relative coordinate *(dx, dy, 0)* is used for positioning instead.

SEE ALSO

moveto(3G)

NAME

moveto – move in absolute space

SYNOPSIS

moveto(x, y [,z])
int x, y, z;

DESCRIPTION

The *moveto* subroutine is called to position to the 2D absolute X, Y coordinates or the 3D absolute X, Y, Z coordinates specified. *X, y, z* are the absolute coordinates. If *z* is not specified the 3-space point *(x, y, 0)* is used for positioning instead.

NOTE

Moveto positions with the homogeneous coordinate (IW) = 32767.

SEE ALSO

move(3G)

NAME

nargs – argument count

SYNOPSIS

nargs( )

DESCRIPTION

Nargs returns the number of actual parameters supplied by the caller of the routine which calls *nargs*.

Arguments are assumed to be of type "int", and hence the count may have to be adjusted if the actual parameters are of a different type. On the PDP-11, *long* counts as two "int's", while *float* and *double* count as four. On the VAX *float* and *double* count as two "int's" and everything else as one.

The former restriction of *nargs* not being able to work with separated I and D space on the PDP-11 has been lifted.

FILES

/usr/lib/libg.a

AUTHOR

Thomas Ferrin, University of California, San Francisco

NAME

nufram – display new frame data

SYNOPSIS

nufram()
and
rstfram()

DESCRIPTION

The *nufram* subroutine is called to initiate the change from displaying old frame data to displaying new frame data. The actual buffer swap does not occur until the appropriate refresh interval has elapsed (see psinit(3G)). *Nufram* returns a non-zero value if data for the new display frame has exhausted available refresh buffer memory (buffer overflow).

Rstfram is used to cancel a partially constructed frame of data. The calling program is suspended until the appropriate refresh interval has elapsed. This call is useful when there has been no change in the current display and the user wishes to suspend program execution until the next display update cycle. This frees the central processor for other functions and is particular efficient in terms of overall system usage.

NAME

pgspsh, pgspop, pgsrd – manipulate file stack

SYNOPSIS FOR C USAGE

```
pgspsh(line);
char *line;

pgspop(line);
char *line;

pgsrd(line, n);
char *line;
int n;
```

SYNOPSIS FOR FORTRAN USAGE

```
call pgspsh(line)
character*(*) line

call pgspop(line)
character*(*) line

call pgsrd(line, n)
character*(*) line
integer n
```

DESCRIPTION

Pgspsh, pgspop, & pgsrd are a set of subroutines for manipulating a "file stack". This stack can be used by more than one program to provide a reasonably straightforward protocol for communicating small amounts of data. This enables multiple programs to run concurrently with one or more programs supplying data and doing calculations. Data on the stack are lines of text; no restrictions other than a maximum length of 150 characters are imposed by the stack structure. It is expected that programs which communicate numerical data via the stack will make use of *sprintf(3S)* or *internal files* (in the case of F77) to set up text lines for the stack.

Pgspsh will take the character string in *line* and push it onto the file */tmp/pgstack??*, creating the file if it does not already exist. If the file does exist, it is opened and then unlinked. A file with the same name is then created and opened for writing. *Line* is written onto the new file and the contents of the original file are then appended.

Pgspop removes the last character string pushed onto the stack (corresponding to the first line in the file). The stack line is returned in *line*.

Pgsrd will find the *n*th element from the top of the stack (the *n*th line from the beginning of the file) and copy it into *line* without changing the stack in any way. If *n* is 1 the the subroutine acts like *pgspop* but without removing the line from the stack.

FILES

/tmp/pgstack??

SEE ALSO

printf(3S)

DIAGNOSTICS

The value -1 and an empty string are returned from *pgspop* and *pgsrd* if there is nothing in the stack.

NAME

pop – pop the matrix stack

SYNOPSIS

pop()

DESCRIPTION

The *pop* subroutine is called to pop the top element of the Matrix Stack into the Picture Processor Transformation Matrix.

SEE ALSO

push(3G), bldcon(3G)

NAME

pot – a simulated potentiometer

SYNOPSIS

int pot(x, y, size, val [,label [,mkob]])
int x, y, size, val;
char *label;
int mkob;

DESCRIPTION

Pot draws a simulation of an interactive potentiometer. This consists of a rectangular area on the CRT screen drawn to represent a slide potentiometer, calibration marks, and a indicator pointer which represents the value to which the pot is set. The user may alter the value of the pot by touching the rectangular area with the tablet cursor. If it is positioned above the centerline and the cursor button is pressed down, the *pot* function returns a positive value; if it is touched below the centerline a negative value is returned. The returned value from *pot* can be used to increment a variable; this variable is typically used as the argument *val* on subsequent calls to *pot*.

Arguments to *pot* include the coordinates of the upper left corner of the potentiometer image area *(x,y)*, and the size of the area. A *size* value of 1 indicates an area of 4K X 1K, and a *size* value of 2 indicates an area of 8K X 2K (in absolute Picture System coordinates). The *val* parameter is a value (range = ±32767) used for positioning the pot's indicator marker ('<-').

Label points to a character string which is displayed in the pot area. This argument is optional; no text is displayed if it is omitted.

The value returned by *pot* is the distance of the tablet cursor from the pot's center-line (when the cursor is within range of the pot and the cursor button is pressed down). Zero is returned when the cursor button is pressed down exactly over the center-line, when the cursor button is not pressed down at all, or when the cursor is not within range of the pot.

The pot and indicator marker become noticeably brighter when the cursor is inside the pot image; this verifies your 'hit'.

The *mkob* parameter is used when the joystick is part of a PS2 display list. It should have a non-zero value when *pot* is called during display list generation, and a zero value during normal display updates. The *label* parameter must be supplied if the *mkob* flag is used. If this parameter is omitted, the entire potentiometer image is redrawn for each frame update.

Pot should be called once in each display loop. It does not call the *tablet* function of the Picture System software itself, but assumes that the user has called *tablet* and that the cursor coordinates are up to date.

SEE ALSO

joystick(3GU), tablet(3G)

BUGS

The intensification of the pot when the cursor is over it depends of the setting of the contrast controls and screen viewport.

NAME

psbuf – set refresh buffer mode

SYNOPSIS

psbuf(status)
int status;

DESCRIPTION

The *psbuf* subroutine is called to set the refresh buffer to single- or double-buffer mode. Once the refresh buffer has been set to either mode, it may be reset at any time to the opposite mode. The user need only call this subroutine if the refresh buffer is to be used in single-buffer mode. *Psinit*, during the initialization process, sets the refresh buffer to the default double-buffer mode.

Status is an integer which specifies the new mode of the Refresh Controller. Valid values for *status* are:

1 = single-buffer mode.
2 = double-buffer mode.

SEE ALSO

psinit(3G)

NAME

pscopy – hardcopy generator for Picture System 2

SYNOPSIS

/usr/lib/pscopy [[-b] filename]

DESCRIPTION

Pscopy is a utility program which generates a hardcopy 'plot' on the Versatec electrostatic printer/plotter from files of display data produced by Picture System programs.

Filename is the data file; "copy.tmp" is the default if none is supplied. This file is automatically deleted after use. The output is designed to fit on a single sheet of Versatec paper. If the **-b** flag is specified, *pscopy* will include a heavy black border, analogous to the hardcopy from *bild(1)*.

Data in the input files is of the format generated by the Picture System *wbtmem(3G)* subroutine, operating in mode 2. The output is produced using the standard Versatec plotting package (version 7).

Writing the data file. The following code can be used:

```
#define BUFSIZ 256
short buf[BUFSIZ];
int size;
int fulsub();
...
wbtmem(2);
...
[display commands go here]
...
rbtmem(buf, BUFSIZ, &size, fulsub);
...
```

FILES

copy.tmp	
/usr/tmp/v*	temporary files for Versaplot

SEE ALSO

fulsub(3GU), bild(1), wbtmem(3G)

BUGS

Tilted characters don't always work right.
There is no way to change plot size.
Due to a Picture System hardware bug, clipped characters are not handled properly.

NAME

pserrs – expand cryptic Picture System 2 error messages

SYNOPSIS

pserrs errno subno [–s [objfil [corefil]]]

DESCRIPTION

Pserrs is a shell-script that provides more meaningful information than the simple

"Error x detected in graphics subroutine y"

message that is produced by the Picture System 2 Graphics Subroutine Package. The message is still terse, but indicates both the type of error and the subroutine name causing the error.

If the -s flag is specified, then a stack trace is produced using *objfil* as the executable program file. The default for *objfil* is *a.out*. *Corefil* is the core image file produced after executing the *objfil*. The default for *corefil* is *core*.

SEE ALSO

Picture System 2 User's Manual, Chapter 6, especially section 6.2 (page 6-75).

NAME

psfini – close all open Picture System 2 files

SYNOPSIS

psfini()

DESCRIPTION

When this routine is called, all open files associated with the Picture System will be closed. This provides a convenient way of insuring that only one process is trying to use the Picture System at any one time (by calling *psfini* in the child process after a *fork)* and also a means to re-initialize the Picture System *(psfini* followed by *psinit).*

FILES

/dev/ps.*

SEE ALSO

psinit(3G)

NAME

psinit – initialize the Picture System 2

SYNOPSIS

psinit([ftime, nrfsh])
int ftime, nrfsh;

DESCRIPTION

The *psinit* subroutine is called to initialize the Picture System 2 hardware and software. The initialization process includes the following:

The Picture System 2 is set to provide refresh of the old frame and timing for frame updates at the intervals specified by the calling argument list.

All variables are assigned their default values. All registers used in the Picture Processor are initialized for two-dimensional drawing mode. The Picture Processor is set to display data unrotated, untranslated, at full brightness, within a viewport which just fills the display screen.

A window is set to include the entire definition space (±32767).

The Refresh Controller is set to double-buffer mode with an initial null frame. The Picture Generator status is initialized to solid line texture and to display characters of .68 cm (.27 inches) character size in horizontal character mode.

Ftime is an integer used to designate the number of 1/120 second intervals per frame refresh. The refresh rates that may be obtained are:

Ftime = 1 for 120 frames per second.
Ftime = 2 for 60 frames per second.
Ftime = 3 for 40 frames per second.
Ftime = 4 for 30 frames per second.
Ftime = 5 for 24 frames per second.

Nrfsh is an integer which specifies the number of frame refreshes which must be completed before a frame update will be recognized. If *Nrfsh* contains a value less than or equal to zero, then frame updates will be allowed upon the next refresh interval after a new frame has been requested. The default values for *ftime* and *nrfsh* are 2 and 4, respectively.

FILES

/dev/ps.*

DIAGNOSTICS

'Warning: frame update rate reset ...' if the user requests a frame update rate greater than 20 frames/second and is not the superuser or a member of the group *rt*.
'Picture System busy' if in use by another process.
'Error x detected in graphics subroutine y' for runtime errors.

SEE ALSO

intro(3G), dowrbuf(3G), pserrs(1G), rtp(3GU)

NAME

psreset – reset the Picture System 2 in time of crisis

SYNOPSIS

psreset

DESCRIPTION

Psreset is designed to be used in those rare instances when a process using the Picture System 2 somehow 'hangs' and cannot be killed by the usual means. It is a practical alternative to rebooting the system.

Psreset sends signal #9 (kill) to whatever process is currently using the Picture System and then forces a reset of the Picture System 2 hardware. In addition, mail is sent to the system administrator detailing selected hardware status and the user id of whoever invoked the command. This information can be useful in tracking down software bugs. Since this command runs as setuid root, even processes owned by other users can be effectively terminated (this is considered a feature).

SEE ALSO

kill(1)

BUGS

Since process ownership is not checked, this command can be abused by the malicious user. Such misuse will be dealt with severly by the administration.

AUTHOR

Thomas Ferrin, University of California, San Francisco

NAME

psstat – report Picture System statistics

SYNOPSIS

psstat [–s] [interval [count]]

DESCRIPTION

Psstat delves into the system and reports on (usually in an iterative fashion) certain statistics kept about Picture System 2 activity. If given a –s argument, it prints the contents of the *cnt* structure, giving the total number of several kinds of PS2 related events which have occurred since the last system reboot. The optional *interval* argument causes *psstat* to report once each *interval* seconds. "Psstat 5" will print what the PS2 is doing every five seconds; this is a good choice of printing interval. If a *count* is given, the statistics are repeated *count* times. The fields are:

Interrupts: detailing the number of different PS2 interrupts per second.

rtc	real time clock
sys	system control
dev	device control
dma	direct memory access

Rqsts: information about the last recorded system & device control requests.

sys	"R,M,H" for refresh stopped, map output stopped & halt requested
dev	"S,K" for stereo image alternator & keyboard

I/O: information about the number of i/o operations and transfer rate.

dma	direct memory access transfers
kbps	dma kilobytes transfered per second
pio	programmed i/o transfers
kbps	pio kilobytes transfered per second

Ps2: breakdown of PS2 usage

map	percentage matrix arithmetic processor busy
idl	percentage ps2 idle

Cpu: breakdown of percentage usage of CPU time

us	user time for normal processes
ni	user time for low priority processes
sy	system time
id	cpu idle

FILES

/dev/kmem, /unix

SEE ALSO

Picture System 2 Hardware Reference Manual

AUTHOR

Thomas Ferrin

NAME

push – push the matrix stack

SYNOPSIS

push()

DESCRIPTION

The *push* subroutine is called to push the current Picture Processor Transformation Matrix onto the Matrix Stack. Note that the Matrix Stack can store a maximum of 8 matrices before overflow.

SEE ALSO

pop(3G), bldcon(3G)

NAME

rdtc – read transformed coordinate data

SYNOPSIS

rdtc(ps2loc, pdploc)
psaddr_t ps2loc;
ps_t *pdploc;

DESCRIPTION

The *rdtc* (ReaD Transformed Coordinates) subroutine is an alternative to the *rbtmem(3G)* subroutine specifically implemented to read back selected transformed coordinate data. By calculating the relative addresses in Picture Memory of only the specific transformed data one is interested in, (X, Y, Z, W) quadruplets of selected coordinates can be quickly and easily extracted from large pictures. Thus, the MAP can be used as a general purpose arithmetic processor for arrays of picture oriented data.

Ps2loc is the location in Picture Memory to begin reading coordinate data (this is the offset from the first datum stored in Picture Memory after *wbtmem* was called and is unusual in that it specifies a WORD [not byte!] offset). *Pdploc* is a pointer to a location in PDP11 memory in which to store the (X, Y, Z, W) values retrieved from the refresh buffer.

SEE ALSO

wbtmem(3G)

NAME

rot – build a rotation matrix

SYNOPSIS

rot(angle, axis)
int angle, axis;

DESCRIPTION

The *rot* subroutine is called to build a rotation transformation based on the angle and axis of rotation specified in the parameter list. The transformation is then concatenated to the Picture Processor Transformation Matrix. *Angle* is an integer which specifies the angle of rotation. The angle is given by dividing a circle into 2**16 equal parts, with zero being equal to zero degrees and 2**15 equal to 180 degrees. Two's complement addition, ignoring overflow, causes the angle to increase counter-clockwise through 360 degrees when viewed along the specified axis in the positive direction. *Axis* is an integer which specifies the axis of rotation. Valid values for *axis* are:

XAXIS (=1) rotation about X axis.
YAXIS (=2) rotation about Y axis.
ZAXIS (=3) rotation about Z axis.

NOTE

The Picture System 2 software is designed for a left-handed coordinate system.

SEE ALSO

scale(3G), tran(3G), getrot(3G)

NAME

rsetps – reset Picture Memory Display Lists

SYNOPSIS

rsetps()

DESCRIPTION

The *rsetps* routine is called to reset the Display List Control Table so display lists may be recreated in Picture Memory. The effect of this call is to cause all current Picture Memory-resident display lists to cease to exist.

SEE ALSO

setps(3G)

NAME

scale – build a scaling matrix

SYNOPSIS

scale(sx, sy, sz [,iw])
int sx, sy, sz, iw;

DESCRIPTION

The *scale* subroutine is called to build a scaling transformation based on the X, Y, Z scaling values specified in the parameter list. The resulting transformation matrix is then concatenated with the current Picture Processor Transformation Matrix. *Sx, sy* and *sz* are the X, Y and Z scaling parameters. The object scaling factor is determined by dividing these parameters by the homogeneous coordinate *iw*. If the *iw* parameter is omitted or given as zero, it is treated as 32767.

As an example, to proportionally scale an object by 1/2 about the x, y and z axes, the *scale* routine could be called with the *sx, sy* and *sz* parameters equal to 16384 (16384/32767 = 1/2). To proportionally scale an object to twice its normal size, the *scale* routine could be called with *sx, sy* and *sz* parameters equal to 32767 and the *iw* parameter equal to 16384 (32767/16384 = 2). To provide for variable scaling which is capable of both increasing and decreasing the size of an object, *sx, sy* and *sz* could be fixed at 8192 and *iw* could be varied from 32767 to 1. This would provide for an object that varied from 1/4 its normal size to 8197 times its normal size.

SEE ALSO

rot(3G), tran(3G), setscl(3G)

NAME

scopes – select Picture System Display

SYNOPSIS

scopes(value)
int value;

DESCRIPTION

The *scopes* subroutine is called to select or de-select the Picture Display to which output will be directed.

Value is an integer which specifies which Picture Display are to be selected or de-selected. *Value* is interpreted as a 6-bit binary value where each bit that is set will select the corresponding scope and each bit that is not set will deselect the corresponding scope. Thus, the value 1 will select scope 0; 2, scope 1; 4, scope 2; 8, scope 3; 16, scope 4; 32, scope 5. The values are additive so that 1+2+4+8+16+32=63 will select all scopes for display.

NAME

setlit – set lights on Function Switches

SYNOPSIS

setlit(n, status)
int n, status;

DESCRIPTION

The *setlit* subroutine is called to set or clear an individual light on the Picture System 2 Function Switches & Lights peripheral, dependent upon the parameters specified to the subroutine. *N* is an integer which specifies the light number that is to be set or cleared. Valid values are:

N = 0-15 for 1 set of Function Switches & Lights
N = 0-31 for 2 sets of Function Switches & Lights
N = 0-47 for 3 sets of Function Switches & Lights
N = 0-63 for 4 sets of Function Switches & Lights

Status is an integer which specifies whether the light is to be set or cleared. *Status* equal to zero clears an individual light; for all other values the light is set.

SEE ALSO

lights(3G), fswitch(3G), iswset(3G)

NOTA BENE

Setlit is acceptable efficiency-wise if one or two lights are being set, but lights(3G) is much more efficient if several lights must be set within the same display loop.

NAME

setps – initialize for Picture Memory Display Lists

SYNOPSIS

setps(limit [,nobs, array])
psaddr_t limit;
int nobs;
int *array;

DESCRIPTION

This routine is called to set the initial lower limit for the Picture Memory refresh buffer. The beginning portion of Picture System memory, up to this limit, is then set aside for storage of structured 'display list' object definitions. This routine must be called AFTER *psinit* is called.

Limit is the lower limit to be established for the refresh buffer. *Nobs* indicates the maximum number of Picture System memory objects. Control information for these objects will be stored in the user supplied PDP-11 memory space pointed to by *array*. Each element is size PSOBSIZE (defined in *ps.h)*. If no user memory space is provided, an array large enough to hold 5 objects will be provided by default.

SEE ALSO

makeps(3G), rsetps(3G)

NAME

sia, left, right – stereo image alternator routines

SYNOPSIS

sia(state)
int state;

right()
and
left()

DESCRIPTION

The *sia* subroutine is called to start or stop the synchronization of the Baush & Lomb stereo image alternator with the Picture System Refresh Controller.

State is an integer which enables or disables synchronization. Valid values for *state* are:

state = 0, synchronization off
state ≠ 0, synchronization on

The *left* and *right* subroutines are called to specify that subsequent data should be displayed for the left or right eye, respectively. When generating pictures, the right eye image should always be displayed prior to the left eye image.

SEE ALSO

siasync(1G)

NAME

siasync – synchronize stereo image alternator

SYNOPSIS

siasync

DESCRIPTION

Siasync facilitates periodic adjustment of the phase relationship between the displayed Picture System image and the Bausch & Lomb stereo viewing shutter. The display consists of a series of letters "LLLLLL RRRRRR" drawn across the face of the CRT screen.

Each letter of the left eye (respectively right eye) pattern is drawn 0.6 ms apart. If the motor housing on the right-side of the stereo viewer is grasped gently but firmly and rotated with respect to the stationary portion of the viewer, phase changes can be seen as a horizontally varying "band" of intensity for each eye's image.

The phase is correct when each eye sees its, and only its, row of characters displayed at equal intensity.

FILES

/dev/ps.map

SEE ALSO

sia(3G)

NAME

speed – set the Line Generator drawing speed

SYNOPSIS

speed(speed)
int speed;

DESCRIPTION

This subroutine sets the Line Generator drawing speed according to the parameter specified.

Speed is an integer which specifies the Line Generator speed. Valid values for *speed* are:

0 = Full speed
1 = 1/2 speed
2 = 1/4 speed
3 = 1/8 speed

NAME

stopob – terminate a Linear Display List

SYNOPSIS

stopob()

DESCRIPTION

The *stopob* subroutine is called to terminate the creation of a Linear Display List previously initiated by a call to the *makeob(3G)* subroutine. The termination of *makeob* mode causes the Picture System 2 software to revert to the normal mode of operation so that all subsequent data 'drawn' will be output to the Picture Processor. A call to the *fulsub* subroutine will also be invoked at this time, providing one was specified by the user in the call to *makeob*.

SEE ALSO

makeob(3G)

NAME

stopps – terminate a Picture Memory Display List

SYNOPSIS

stopps()

DESCRIPTION

This routine is called to terminate the creation of a Picture Memory-resident display list initiated by a previous call to the *makeps, maksps* or *apndps* subroutines. The termination of *makeps* mode causes the Picture System software to revert to the normal mode of operation so that all subsequent data 'drawn' will be output to the Picture Processor.

SEE ALSO

makeps(3G), apndps(3G)

NAME

stopwb – terminate write back mode

SYNOPSIS

stopwb()

DESCRIPTION

The *stopwb* subroutine is called to terminate the write-back to memory mode of operation initiated by a previous call to the *wbtmem(3G)* subroutine. The termination of *wbtmem* mode causes the Picture System 2 software to revert to the normal mode of operation so that all subsequent data output to the Picture Processor will be transformed and output to Picture Memory for display. A call to the *fulsub* subroutine will also be invoked at this time, provided one was specified by the user in the call to *wbtmem*.

SEE ALSO

wbtmem(3G)

NAME

subps – display list structuring

SYNOPSIS

subps(name)
int name;

DESCRIPTION

This routine causes a subroutine jump (PUSHJ) to object *name*. The jump instruction is placed into the display list currently being created in Picture Memory. This serves to introduce a structure into an otherwise linear list. Hence, a display list generated by the *makeob* routine is termed a 'Linear Display List', while one created by *makeps* is sometimes called a 'Structured Display List', even though it need not contain structuring.

Note that a call to this subroutine is only valid when the Picture System software is in *makeps* mode. Also, display list *name* must already reside in Picture Memory.

Subobjects may be nested up to 16 levels deep (no overflow checking is done).

Name is the object identifier.

SEE ALSO

makeps(3G)

NAME

tablet – retrieve data tablet cursor position

SYNOPSIS

tablet([x, y, pen])
int *x, *y, *pen;

DESCRIPTION

Tablet retrives the current cursor switch status and position information from the data tablet. *X* and *y* are addresses of integers which are updated with the current pen position. *Pen* is the address of an integer which is updated with the current pen information. Bit 4 will be set if the pen is down and bits 2-0 will be zero if the pen is within proximity of the tablet surface. All pen bits are mnemonically defined in *ps.h*. If the *x, y* and *pen* arguments are omitted, the default external variables *_ix, _iy* and *_ipen* will receive the tablet information.

Tablet returns a non-zero value if the cursor sense switch is currently depressed.

SEE ALSO

cursor(3G), ispchd(3G), /usr/include/ps.h

BUGS

Changing the tablet size from the normal 11"x11" model requires changing parameter values in the device driver.

NAME

text – display text

SYNOPSIS

text([count,] string)
int count;
char *string;

DESCRIPTION

The *text* subroutine is called to display the text string specified in the parameter list. The display of the text will be from the current position and at the intensity associated with the last information displayed on the screen. *Psinit* initializes the character status; it may be updated by calling the *charsz* subroutine. *Count* is an optional argument which specifies the number of characters to be displayed. *String* is a pointer to the array of characters to be displayed.

If *count* is omitted, then the characters from the beginning of *string* up to the first null or non-ASCII character will be displayed.

SEE ALSO

charsz(3G), psinit(3G), getchr(3G)

NAME

tran – build a translation matrix

SYNOPSIS

tran(x, y, z [,w])
int x, y, z, w;

DESCRIPTION

The *tran* subroutine is called to build a translation transformation based on the X, Y, Z translational values specified in the parameter list. The transformation is then concatenated to the Picture Processor Transformation Matrix. *X*, *y* and *z* are the scaled translation values. *W* is the factor used to scale the translational values. If the scale factor is omitted or given as zero, it is treated as 32767.

SEE ALSO

rot(3G), scale(3G), gettrn(3G)

NAME

transpose, trpose – transpose of a Picture System matrix

SYNOPSIS

transpose(func, mat);
int func;
int mat[4][4];
and
trpose(mat, xtrans);
int mat[4][4], xtrans[4][4];

DESCRIPTION

Transpose calculates the transpose of a Picture System transformation matrix supplied in *mat* and passes the result to *bldcon(3G)* along with the specified function type *func*.

Trpose calculates the transpose of *mat* and places the result in *xtrans*.

If the matrix *mat* is a strict rotation operator, then the transpose of *mat* is equal to the inverse rotation.

SEE ALSO

Evans and Sutherland Picture System II User's Manual, Section 4.2.3

NAME

txture – set line texture

SYNOPSIS

txture(status [,cont])
int status, cont;

DESCRIPTION

The *txture* subroutine is called to set the Line Generator status such that all subsequent lines will be drawn in the selected mode. *Status* specifies the line mode to be selected:

0 = solid lines.
1 = lines consisting of short dashes.
2 = lines consisting of medium-short dashes.
3 = lines consisting of medium-long dashes.
4 = lines consisting of long dashes.
5 = lines consisting of long-short dashes (centerline).
6 = lines consisting of long-short-short dashes.

Cont, if specified and non-zero, enables continuous texture mode for the Line Generator (see Picture System 2 reference manual, section 2.4.3b).

NAME

unit – load a unit matrix into the Picture System MAP

SYNOPSIS

unit ([iw]);
int iw;

DESCRIPTION

Unit replaces the current transformation in the Picture System's Matrix Arithmetic Processor (MAP) with a unit matrix. The level of the MAP stack is not affected and the transformation in the MAP when *unit* is called is lost.

If *iw* is specified, it is used as the value of the diagonal elements in the transformation. If it is omitted, the value 16384 is used.

SEE ALSO

bldcon(3G)

NAME

vv3, vxv3 – dot and cross product of two 3-dimensional vectors

SYNOPSIS FOR C USAGE

double vv3(a,b);
double a[3],b[3];
and
vxv3(c,a,b);
double a[3],b[3],c[3];

SYNOPSIS FOR FORTRAN USAGE

real vv3(a,b)
real a(3),b(3)
and
call vxv3(c,a,b)
real a(3),b(3),c(3)

DESCRIPTION

Vv3 returns the inner (dot) product of two three-dimensional vectors.

Vxv3 calculates the cross-product of the three-dimensional vectors *a* and *b*, and places the result in vector *c*.

SEE ALSO

dist3(3GU)

NAME

vwport – set screen viewport

SYNOPSIS

vwport(l, r, b, t, h, y)
int l, r, b, t, h, y;

DESCRIPTION

The *vwport* subroutine is called to set a viewport as specified by the calling parameters. *L, r, b* and *t* are respectively the left, right, bottom and top boundaries. The normal range for these values is -2048 to 2047. *H* and *y* specify the display intensity are at hither and yon clipping planes. The normal range for these values is 255 for full intensity to 0 for no intensity.

SEE ALSO

window(3G), psinit(3G)

NAME

wbtmem, rbtmem – write back to memory

SYNOPSIS

wbtmem(type)
int type;
and
rbtmem(array, size, lenp [,fulsub])
ps_t *array;
int size;
int *lenp;
int fulsub();

DESCRIPTION

The *wbtmem* (Write-Back To MEMory) subroutine is called to initiate a mode of operation whereby all data written out to the Picture System 2 is formatted in a manner different than the usual 'display' mode of operation. A subsequent call to *rbtmem* will then store this transformed data in a user supplied buffer and restore the Picture System to its status just prior to the *wbtmem* call. (The routine *stopwb* may also be called to restore the original status any time write-back mode is active.)

Type is an integer which specifies the type of data transformation that is to occur. Valid values are:

1 = data transformed only.
2 = data transformed and clipped.

Array is the address of a user supplied buffer which will hold the write-back data. *Size* is an integer which specifies the number of elements in the buffer. *Lenp* is a pointer to an integer variable where the number of buffer elements actually used will be maintained. *Fulsub* is the address of a subroutine to be called if the available buffer space becomes exhausted.

If supplied, the C calling sequence will be:

fulsub(array, len, flag);

When called, the user's subroutine should empty the write-back buffer and then return. *Flag* will be non-zero on the last call to *fulsub*, that is, when the contents of the write-back area of PS2 memory have been exhausted. When *wbtmem* is called, and then other graphic subroutines are subsequently called, the data is transformed by the MAP according to *type* and then stored in Picture System 2 refresh buffer. A call to *rbtmem* will then transfer this data to the user buffer area in one of two formats. The formats are:

For *type* = 1 and all 'FSM2' values:

X-coordinate (1 word)
Y-coordinate (1 word)
Z-coordinate (1 word)
W-coordinate (1 word)
<repeat>

For *type* = 2:

Command Code (1 word)
Data (3 words)
<repeat>

Where *command code* is:

0 = MOVETO
1 = DRAWTO
2 = TEXT
3 = STATUS DATA
-1 = End Of Frame

and 'data' is:

(For command codes 0 and 1)
X-coordinate (1 word)
Y-coordinate (1 word)
Z-coordinate (1 word)

(For command code 2)
character 1 (1 byte)
character 2 (1 byte)
character 3 (1 byte)
character 4 (1 byte)
--- 0 --- (1 word)

(For command code 3)
Line Generator Status (2 words)
--- 0 --- (1 word)

For text strings which are longer than 4 characters, additional command code and data sequences will be output. The end of text is indicated by a null byte or a command code != 2.

SEE ALSO

rdtc(3G), fulsub(3GU), pscopy(3GU), Picture System 2 reference manual (section 2.3.3)

BUGS

Due to a hardware design botch, the sign of the transformed data may not be correct. To correctly reflect the true sign bit the transformed data must also be normalized; however, this gives inconsistent results for different values of the 'FSM2' bit field of the draw command. In practice, few sign-bit errors have actually been encountered.

NAMES

window – set window

SYNOPSIS

window(l, r, b, t [,w])
int l, r, b, t, w;
and
window(l, r, b, t, h, y [,eye [,w]])
int l, r, b, t, h, y, eye, w;

DESCRIPTION

The *window* subroutine concatenates a two-dimensional or three-dimensional windowing transformation to the Picture Processor Transformation Matrix. This subroutine can be used to perform two-dimensional windowing, orthographic projection or a true perspective transformation of data. The windowing transformation is constructed from the arguments specified in the parameter list.

L, r, b and *t* are respectively the left, right, bottom and top scaled window boundaries in definition space coordinates. *H* and *y* are the hither and yon boundaries. For two-dimensional windowing, the window front, or hither, boundary is 0; the window rear, or yon, boundary is equal to *w*. For three-dimensional windowing, if *y* equals *w* the yon boundary is positioned at infinity on the side of the hither clipping plane opposite the eye so that no yon clipping will be performed. *Eye* is an integer which, if specified, is the scaled Z position of the eye. If this parameter is omitted or equals *w*, the eye is positioned at minus infinity which produces an orthographic view of the data. *W* is an integer used to scale the window boundaries and eye position. If the scale factor is omitted or given as zero, it is treated as 32767.

SEE ALSO

vwport(3G), psinit(3G)

NAME

xerrors – expanded format for error printout

SYNOPSIS

xerrors()

DESCRIPTION

This routine enables the expanded form of error printout as described in *intro(3G)*.

SEE ALSO

intro(3G), pserrs(1G)